Harry Zhu

CS150

Project 3 Report

Introduction

This project is designed to read a text file and creates a book index of the text file mapping all the English words from the book to the index. Compare to a typical book index, which appears at the end of a book and lists the page number for all the important words in the book, this book index lists all the English words and the line number where they appear, since there is no page numbers in a text file. For the purpose of the program, all words listed in the Index are converted to lower case, and certain words with symbols are ignored. Only words from a certain given English dictionary are considered English words. For the data structure of the index, three are implemented for comparison: Sorted List, Tree Map, and Hash Map. The project demonstrates the familiarity of the author to all three data structures, with achieving the exact same goal with all three data structures.

Approach

One important purpose of the lab is to demonstrate the author's familiarity with the map data structure, which can be implemented with two map utilities from java, and also with array lists.

To make sure only English words are added into the indices, one dictionary class is created, which contains an array list of Strings, all the words in the dictionary. In the constructor, the class scans one file, as it splits the lines from the file and add all the words into the array list. One public method is also created, which uses a binary search to find whether any given word exists in the dictionary.

The most important part creating a book index is three index classes which uses three different data structures, but all implement one abstract class, Indices. The abstract classes Indices has three field variables which will be used by all three subclasses. One is an dictionary

which will be used to check whether a word is an English word, one is an array list which will be used to store all the lines that is scanned from a text file, and one is an integer, wordcount. The abstract class also has two abstract methods, which are both implemented by its subclasses. One is *addToIndex* which takes an string parameter for file name, and adds all the words from the file to the index container, and the other is *printAll*, which also takes a String for filename and prints the entire index to that file. The index container, however, is not a part of the abstract class since all three subclasses will be using different data structure to contain the index.

The first index class, SortedListIndex, uses an array list to store the Index. To implement such structure, an node class is first created to simulate the structure of each entry in a map, called Entry. The entry class has two field variables. The first one, key, is a string, which stores the key English word of the entry, and the second, values, is a tree set of Integers, which stores all the line numbers of the word. A tree set is used here so that the values will be easier to be accessed with ascending order. The entry class has four public methods. The first one, addValue takes an integer and adds it to the tree set *values*. The *compareTo* method compares the entry object with another entry object by comparing their key word, so that the array list can be sorted alphabetically, and there will not be two entry with the same key words. The third method equals achieves the same goal. And the last method, *toString,* returns the key and all its line numbers in a string.

Looking at the main body of the *SortedListIndex* class, it uses an array list of entry object to implement a map. In the *allToIndex* method, it scans all lines of a file into an array list, which preserving the line number of each line in its index number. After the scanning is complete, each line is taken out from the array list, split into an array of string. Then each String is taken out of the array, and the program first checks whether the word is in the dictionary, then checks if the word is already in the index with a binary search. If the said entry already exists, the entry is then taken out of the index and the line number will be added to the entry, and if not, the new entry with the word and line number will be added into the correct position of the index. At the end of the method, total time of executing the method is taken and returned. For the *printAll* method of the *SortedListIndex* class, the array list index is simply traversed from the

head to tail, and each entry printed to a file, since the array list is already sorted. And an executing time is returned.

The *HashmapIndex* class and the *TreemapIndex* shares an identical *addToIndex* method. Both classes implement map data structure, with keys being the String of key words, and value being a tree set of Integers which contains all the line numbers a key word exists in. Firstly, it adds all lines into the array list lines like *SortedListIndex* class does. The way it distinguishes itself from the same method in *SortedListIndex* class is that an entry object is not needed. When a new entry is needed, it simply uses a *put* method of the map class. And when a key already exists in the map, the program takes out the tree set mapped to the key and adds a line number to the tree set.

The *printAll* method of the two classes, however, are slightly different. In the tree map class, print method extract all the keys into a set, get all values from the map by traverse the set, and prints all values into a file, since a tree map is already sorted. The printed result will be alphabetical. For the hash map index class, however, print method require an additional step of sorting the extracted keys before traversing the set, which the extracted keys will not be sorted.

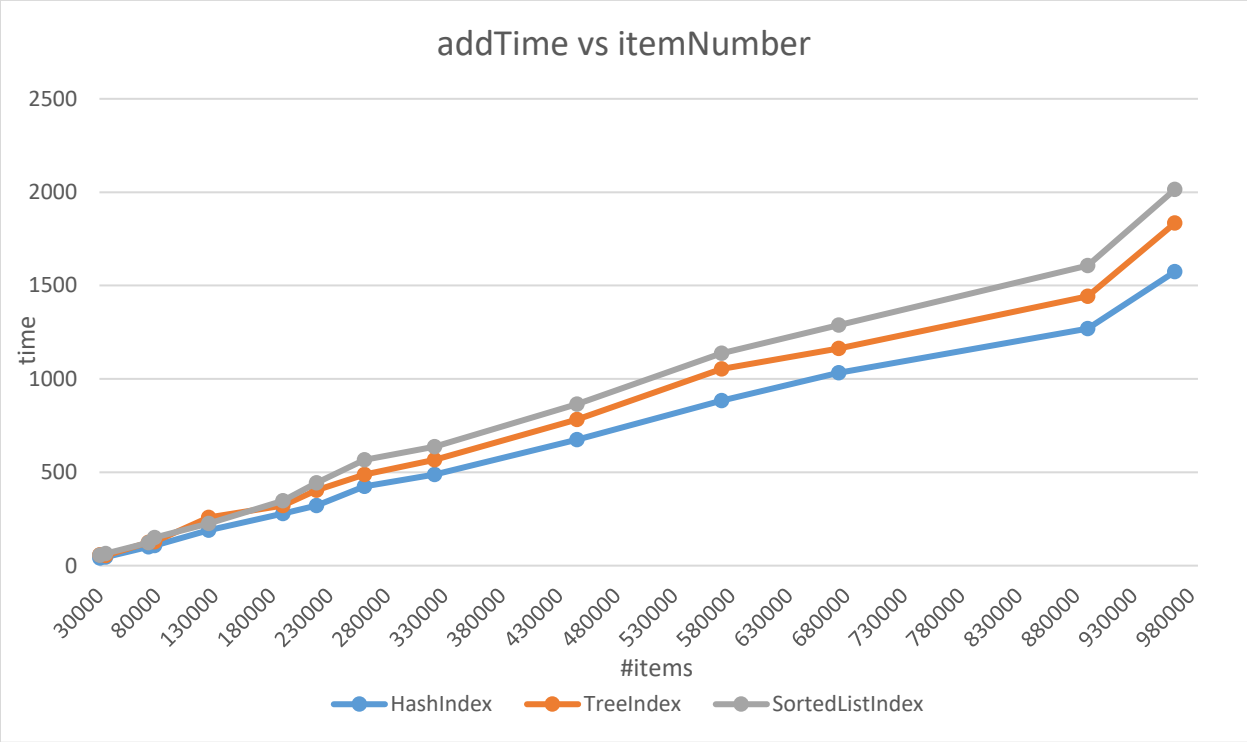
Such description concludes the main structure of the project.

Methods

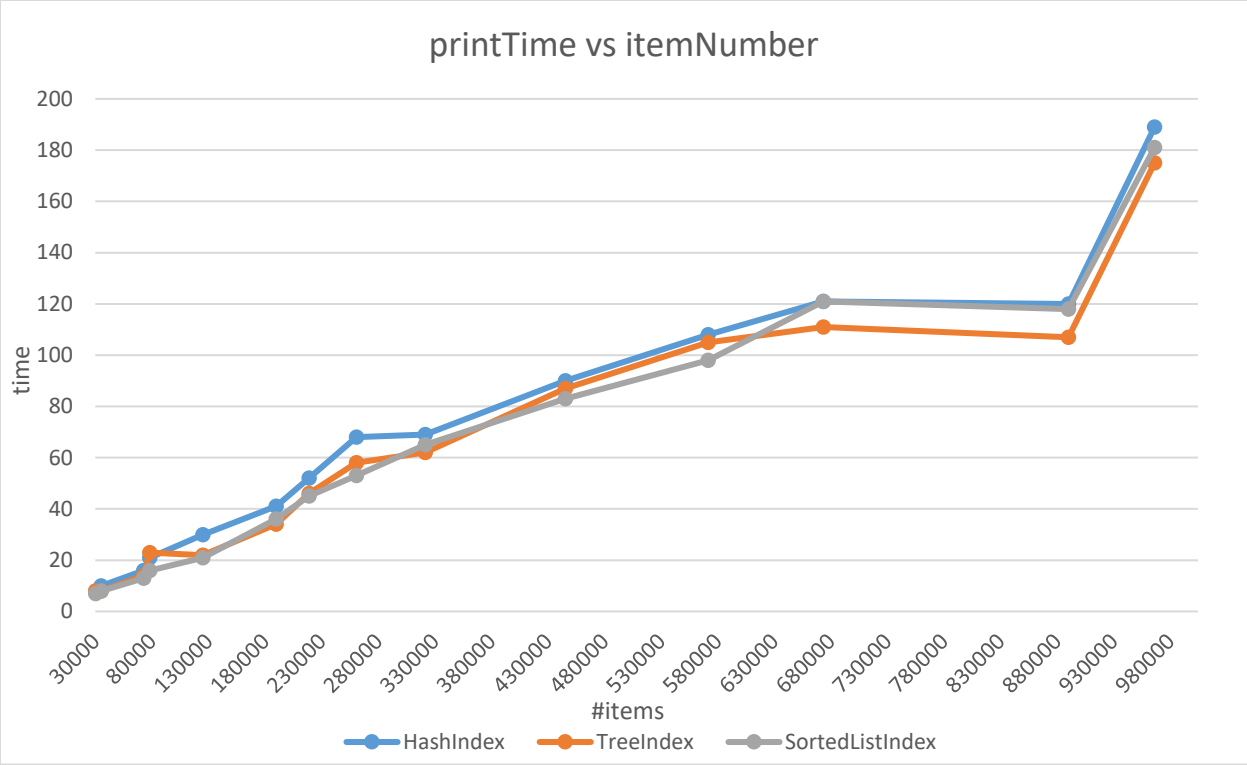
To conduct the proper experiment of the project, and in order to compare the three data structures implemented in the project, first a few e-books of very different sizes are downloaded. A total of 14 books are downloaded with world count spread from thirty thousand words to a hundred thousand words.

For the experiment, each file is imported into each indices five times after each index is cleared from each trial, and then printed into a corresponding file in the output folder, while an average of the five times is recorded also in corresponding files.

Data Analysis



The graph above shows the adding time vs. item number relation for all three indices. It clearly shows a speed difference between the three, as hash map runs faster than tree map, which also runs faster than a sorted list. Such behavior is expected, as adding in a hash set has an time complexity of O(1), while tree set(red black tree) has time complexity of O(log n), and array list being O(n). As the horizontal vertex is scaled properly, all three data structures show an almost linear relation between the two because a series of complexities in the program. While search time and add time is mixed with add time into a tree set which all three classes uses to store line numbers, a time of binary search is also added each time a word is being added, since only English words can be added into the set.



Above graph shows the printing time vs. item number relation for all three indices. It clearly shows a speed difference between the three, as hash index is slower than tree index, which is then slower than a sorted list, although at very large quantities of storage, sorted list index seems to slow down. Hash map is the slowest since the print method includes an additional line of sorting the keys extracted, as they are not sorted after being extracted from the hash map. Tree map shows slightly slower time than sorted list because an extra step of extracting the key set is still required for printing.

Conclusion

The project thoroughly examines the methods of processing large quantity of data by mapping the position of each data. It also shows the performance difference between implementing the map structures with three data structures. After the experiments from the project, a much more all-around understanding of maps is established.

Reference

The CS-150 Lecture Notes for the data structures:

Ge, Xia. “Lecture Note 10 Sets and Maps & Note 3 Lists.” Topic 2. Lafayette College, Easton. Apr 15 2020 Presented. Lecture.

The CS150 Textbook:

Weiss, Mark Allen. Data Structures & Problem Solving Using Java. Langara College, 2017.The ArrayList Class Java API:

ArrayList (Java Platform SE 8), 20 Apr. 2020, docs.oracle.com/javase/8/docs/api/ja va/util/ArrayList.html.

The TreeMap Class Java API:

TreeMap (Java Platform SE 8), 20 Apr. 2020, docs.oracle.com/javase/8/docs/api/ja va/util/TreeMap.html.

The HashMap Class Java API:

HashMap (Java Platform SE 8), 20 Apr. 2020, docs.oracle.com/javase/8/docs/api/ja va/util/HashMap.html.

The Scanner Class Java API:

Scanner (Java Platform SE 8), 20 Apr. 2020, docs.oracle.com/javase/8/docs/api/ja va/util/Scanner.html. 7. The PrintWriter Class Java API:

PrintWriter (Java Platform SE 8), 20 Apr. 2020, docs.oracle.com/javase/8/docs/api/java/util/PrintWriter.html.

The sorting method from Java.Collections:

Collections (Java Platform SE 8), 20 Apr. 2020, docs.oracle.com/javase/8/docs/api/java/util/Collections.html.

The Binary Search method from Java.Collections:

Collections (Java Platform SE 8), 20 Apr. 2020, docs.oracle.com/javase/8/docs/api/java/util/Collections.html.