Data Structures Report

Coursework 1 — Word Index

Hash table and Linked List based Map Implementations

Name: Jose Carlos Cerqueira Fernandes

Student Number: H00324200

Contents

System’s Design

The program was given with the capability to read multiple files, word by word. The object word is used when reading the file to allow word manipulation, it holds a position and its respective word string. The main interfaces necessary for the execution of this program were ready to implement. Some of the classes were already implemented and the “WordIndex” file was partially implemented.

This program intends to compare the “HashWordMap” and the “ListWordMap” which had to implement the “IWordMap” that implements the necessary functions. This interface casts the map running to only execute functions declared directly by the interface itself and indirectly inherited by the object class. This is very important to reduce the interdependence between the “WordIndex” module and the map (reduces coupling) and it increases the focus and the purpose of the map classes (increases cohesion).

The Maps manipulate entries and as such, the “Entry” generic interface was created. This interface follows the abstract data type definition. This interface might appear to be redundant but it can help to increase flexibility in the future if it is allowed to modify the current interfaces as a result of this program being required to handle other types of information. Otherwise, the hash and list maps would be implemented generically.

The “WordEntry” class implements the Entry interface and it is manipulated by the linked and hash-based maps. This class defined its entry key as the word itself and the value as a resizable array (Java’s Arraylist) holding “IPosition” values. Besides, the “WordEntry” has the duties of put and removing positions, to return the array length and the Iterator. The “IPosition” interface holds the line integer value and the file name string, to store where the word appears. Additionally, the “IPosition” is used multiple times in the “WordEntry”, “IWordMap” and the “WordIndex” to cast the “WordPosition” class into an instance that only returns the file name string and the line integer.

The “IMap” interface also throws a “WordException” error class. This class is inherited from Java’s “Exception” class and it can be caught by the try-catch block. The “WordException” is thrown when an entry is not found or if a word is not associated with the position. This Exception is very useful because it informs if an entry associated with the position is present on the map. This allows counting added and removed entries during the execution of “WordIndex”.

The ”ListWordMap” is an implementation of “IWordMap” that stores the entries using the Java’s “Linked List”, which implements the Java’s “List” interface that stores an ordered collection of elements. The “Linked List” does not store the elements contiguously but uses instead the memory addresses to chain each other. This data structure is very efficient doing operations such as deleting, inserting and updating (time complexity O(1)) but the search and adding are costly (time complexity O(n)) because the elements are not stored contiguously so there are no indexes to find immediately the correct position, since it is an ordered list.

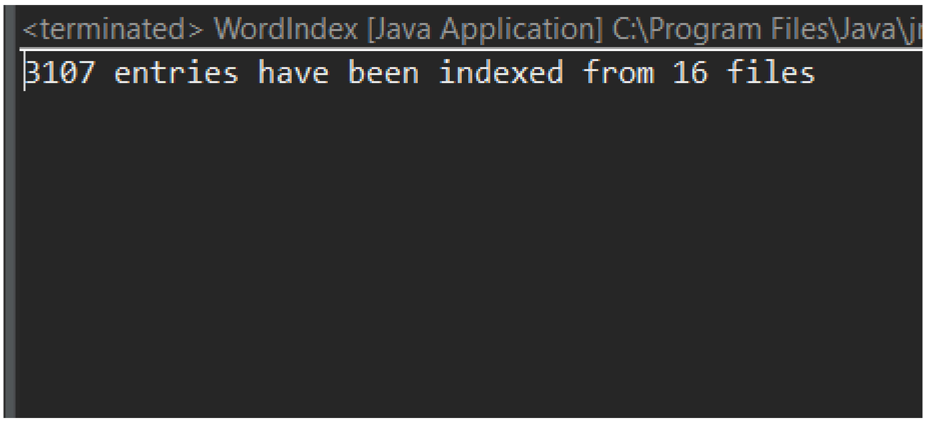
The “HashWordMap” is based on a pure array of Word Indexes. Due to this implementation, the operations removing, searching and updating are very efficient and are constant in the time complexity (time complexity O(log n)). The adding operation is also very efficient in almost every scenario (approximate average time complexity O(log n)) but it is worst-case time complexity is (time complexity O(n)) because it relies on a specific maximum load factor (number entries divided by the array length) specified during running time. If the array load factor the array is overflooded, the array needs to be expanded and the existing entries need to be hashed and added yet again. This implementation of a hash map has the limitation of not condensing the array if the load factor is too low. If this implementation included this feature, the removing operation would not be constant anymore, even having mostly the same performance (approximate average time complexity O(log n)), but the worst case would be the same as the adding operations as it would need to rehash and re-add the existing entries into a new smaller array. This limitation is important to consider in the case it is necessary to remove a large bulk of entries (example: removing many files), and the array is not mostly unused costing too much memory for data being stored. Additionally, the “hashCode” function is limited to the polynomial value of the Java’s long integer absolute maximum (9,223,372,036,854,775,808), so if the string is long it will return a value which will be out of the bounds of this data type.

The ”search” requires the map to get the all the positions (using an iterator) for a specific word entry key to then sort the files by the number of positions, by decrescent order, of the given word entry key, including the lines the word appears in each file. The “WPositionsFile” class holds the file name and positions per each file, the intent is to group the positions by file since the iterator only contains the position of a single entry. This class is then grouped in a pure array, and the comparator “ComparatorSortByPositionsAmount” class is used to compare each “WPositionsFile” in descending order.

Output Examples

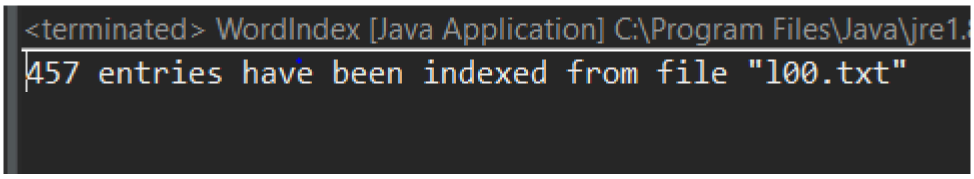
HashWordMap

“addall”



```
<terminated> WordIndex [Java Application] C:\Program Files\Java\j
3107 entries have been indexed from 16 files
```

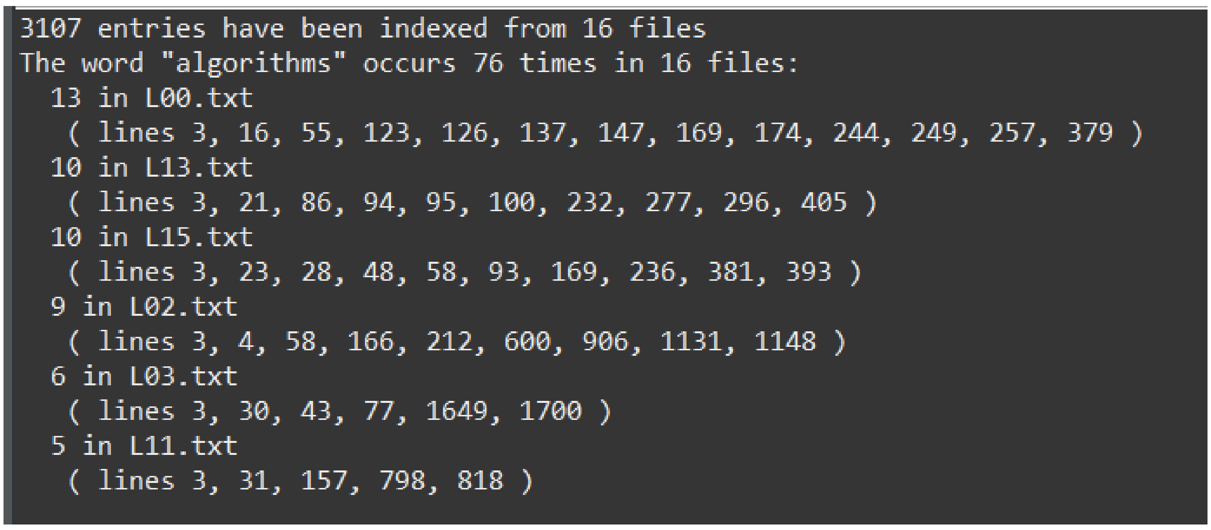
“add l00”



```
<terminated> WordIndex [Java Application] C:\Program Files\Java\jre1.
457 entries have been indexed from file "l00.txt"
```

“search 6 algorithms”

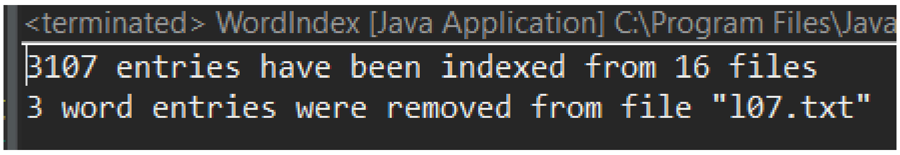
It requires to run the command “addall”



```
3107 entries have been indexed from 16 files
The word "algorithms" occurs 76 times in 16 files:
  13 in L00.txt
   ( lines 3, 16, 55, 123, 126, 137, 147, 169, 174, 244, 249, 257, 379 )
  10 in L13.txt
   ( lines 3, 21, 86, 94, 95, 100, 232, 277, 296, 405 )
  10 in L15.txt
   ( lines 3, 23, 28, 48, 58, 93, 169, 236, 381, 393 )
  9 in L02.txt
   ( lines 3, 4, 58, 166, 212, 600, 906, 1131, 1148 )
  6 in L03.txt
   ( lines 3, 30, 43, 77, 1649, 1700 )
  5 in L11.txt
   ( lines 3, 31, 157, 798, 818 )
```

“remove l07”

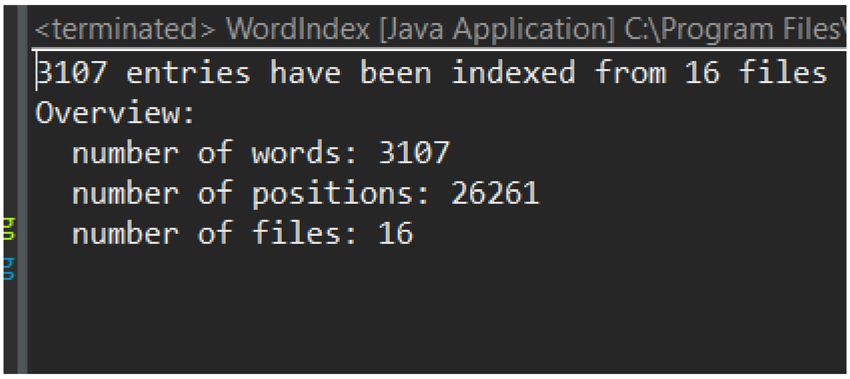
It requires to run the command “addall”



```
<terminated> WordIndex [Java Application] C:\Program Files\Java
3107 entries have been indexed from 16 files
3 word entries were removed from file "l07.txt"
```

“overview”

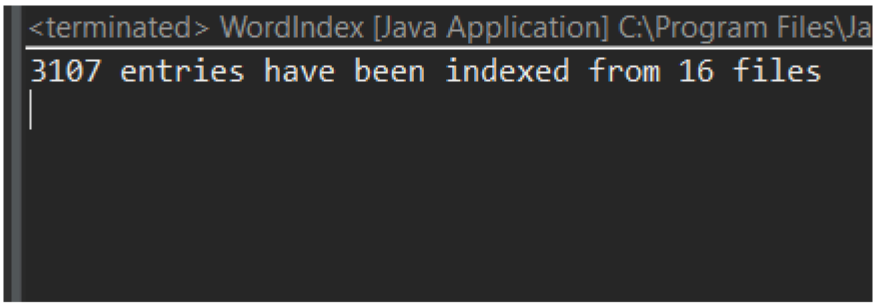
It requires to run the command “addall”



```
<terminated> WordIndex [Java Application] C:\Program Files\
3107 entries have been indexed from 16 files
Overview:
  number of words: 3107
  number of positions: 26261
  number of files: 16
```

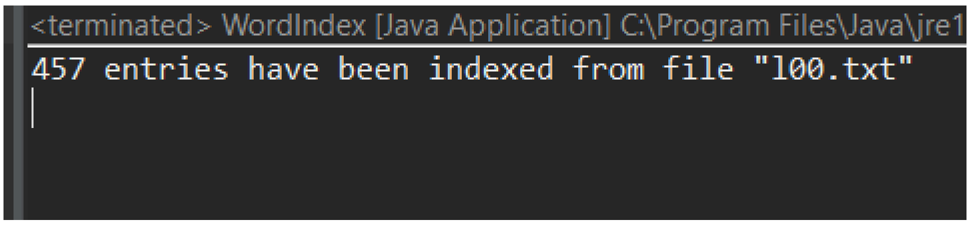
ListWordMap

“addall”



```
<terminated> WordIndex [Java Application] C:\Program Files\Ja
3107 entries have been indexed from 16 files
```

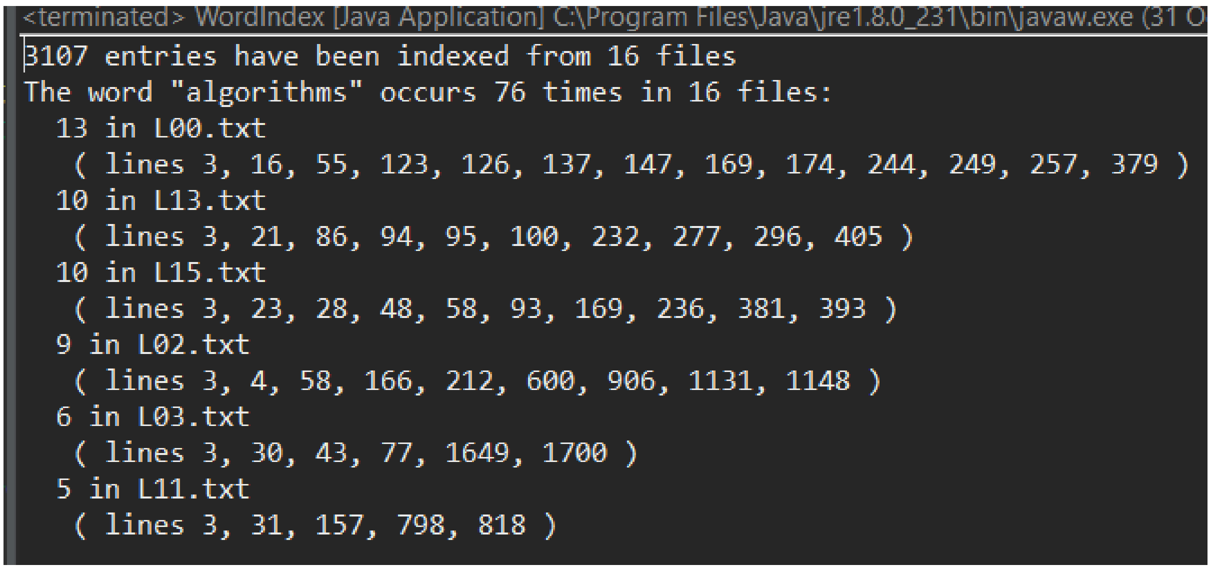
“add l00”



```
<terminated> WordIndex [Java Application] C:\Program Files\Java\jre1
457 entries have been indexed from file "l00.txt"
```

“search 6 algorithms”

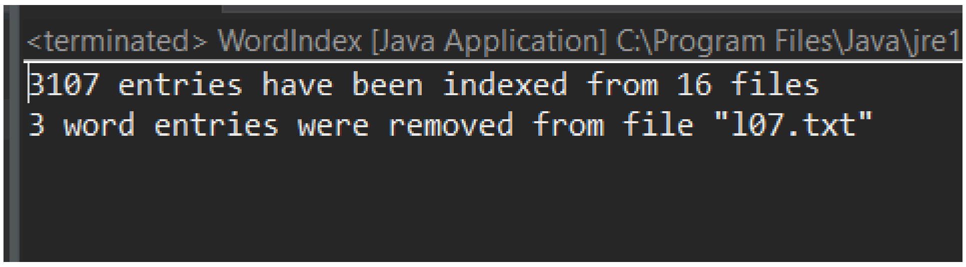
It requires to run the command “addall”



```
<terminated> WordIndex [Java Application] C:\Program Files\Java\jre1.8.0_231\bin\javaw.exe (31 O
3107 entries have been indexed from 16 files
The word "algorithms" occurs 76 times in 16 files:
  13 in L00.txt
   ( lines 3, 16, 55, 123, 126, 137, 147, 169, 174, 244, 249, 257, 379 )
  10 in L13.txt
   ( lines 3, 21, 86, 94, 95, 100, 232, 277, 296, 405 )
  10 in L15.txt
   ( lines 3, 23, 28, 48, 58, 93, 169, 236, 381, 393 )
  9 in L02.txt
   ( lines 3, 4, 58, 166, 212, 600, 906, 1131, 1148 )
  6 in L03.txt
   ( lines 3, 30, 43, 77, 1649, 1700 )
  5 in L11.txt
   ( lines 3, 31, 157, 798, 818 )
```

“remove l07”

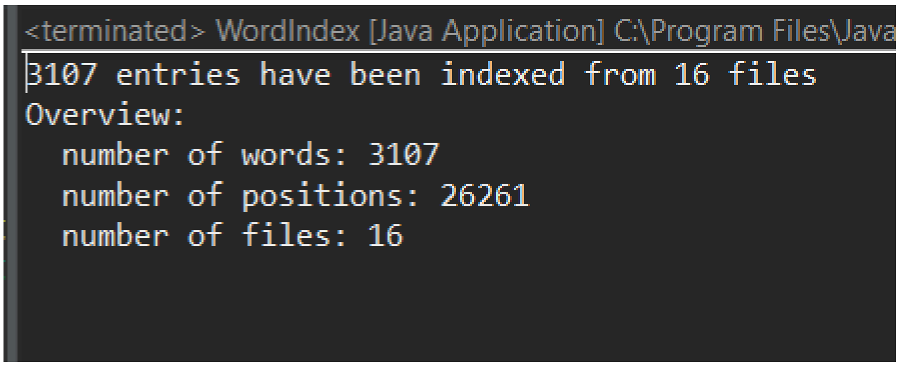
It requires to run the command “addall”



```
<terminated> WordIndex [Java Application] C:\Program Files\Java\jre1
3107 entries have been indexed from 16 files
3 word entries were removed from file "l07.txt"
```

“overview”

It requires to run the command “addall”



```
<terminated> WordIndex [Java Application] C:\Program Files\Java
3107 entries have been indexed from 16 files
Overview:
  number of words: 3107
  number of positions: 26261
  number of files: 16
```

Chart runtime comparison Comparison

Command	HashWordMap (milseconds)	LinkedListMap (milseconds)
“addall”	1116	55003
“add l00”	43	56
“search 6 algorithms”	3	7
“remove l00”	2	69
“overview”	15	15823