CMSC 676 Information Retrieval Project Phase-3

Submitted By- Vinayak Salavi (TL26978)

Implementation

Programming language used: Java

Parser used: Jsoup

Approach

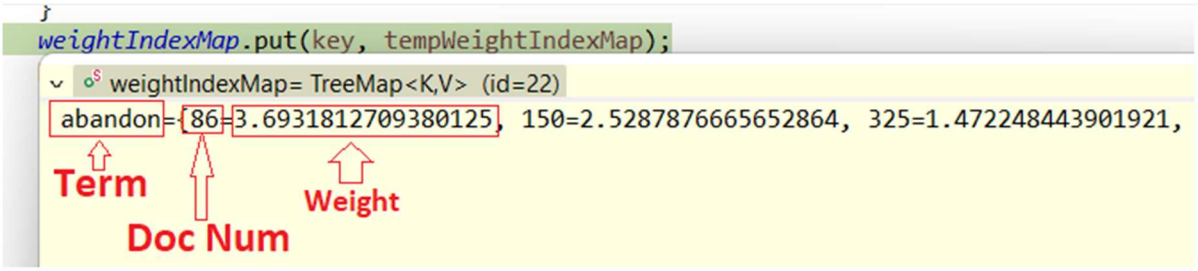
To proceed with the third phase I have updated my existing implementation.

- For phase 2 I had used SOA formula to calculate the term weights.

```
int Cij = map.get(docID);// C(i,j)count of word in currenct doc
// D(i) = wordCountPerDocArr[i] = total words in current doc
// SOA formula for term frequency
tf = Cij / (float) (Cij + (k * wordCountPerDocArr[docID - 1]) / averageDoc);
// D = 503 ,
idf = Math.log(totalDoc / (double) DFj);
weight = tf * idf;
```

  Also, for this phase to create the inverted index, I used a treemap of type Map<Word, Map<DocID, Count>> to store the word and corresponding count of the word along with the document ID.
- To proceed with phase 3 I used a structure similar to the inverted index to store the weights. `TreeMap<String, TreeMap<Integer, Double>> weightIndexMap`
  The use of TreeMap saved me an additional round of sorting as Treemap stores values sorted based on keys.

- PFB screenshot of debugging for term-doc-weight data



- Although I have stored the weights of terms in a different map, to save the processing time (i.e. instead of an additional iteration over the weight map) for the creation of the dictionary and posting file, I'm using the same loop which was used to calculate the weights.

- To explain it further, I've used the **term at a time** approach instead of going document by document. While iterating over the inverted index created, I've created both posting and dictionary files. These are being continuously updated as the weights for the terms are being calculated. After calculating the weight for a key, it's been stored in a temporary map along with the corresponding document id.
- The weight calculated along with the respective doc id is also written in the posting file created. Then the temporary map created in the previous step is added as a value to the weight map.
- The dictionary file is also created in the same loop while iterating over the inverted index. To keep the track of the line number I have created a variable `currLineNumber.` This is incremented by the size of the map<docId-count>. (if iterated over the weight map it would be the size of the posting list.) This can be represented as `currLineNumber += indexMap.get(key).size();`
- All output files will be stored under a newly created repository output_files.

Steps to execute:

- Compile
  - javac -cp ".\jsoup-1.14.3.jar;" .\FileParser.java

- Run – Program takes 2 command line arguments: 1. Path to input files 2.Path to store output files
  - java -cp ".\jsoup-1.14.3.jar;" FileParser "Input_File_Path" "Output_File_Path"

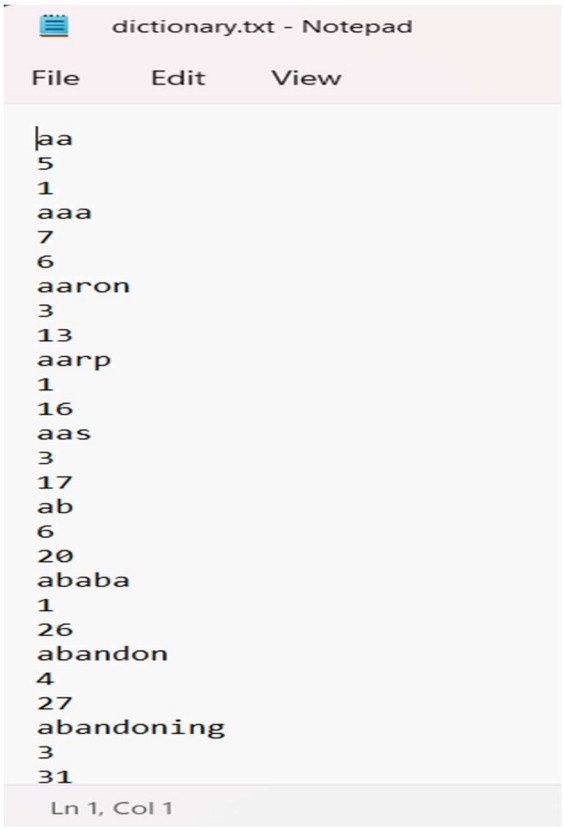
  e.g. java -cp ".\jsoup-1.14.3.jar;" FileParser "D:\IR\Assignment 1\files"

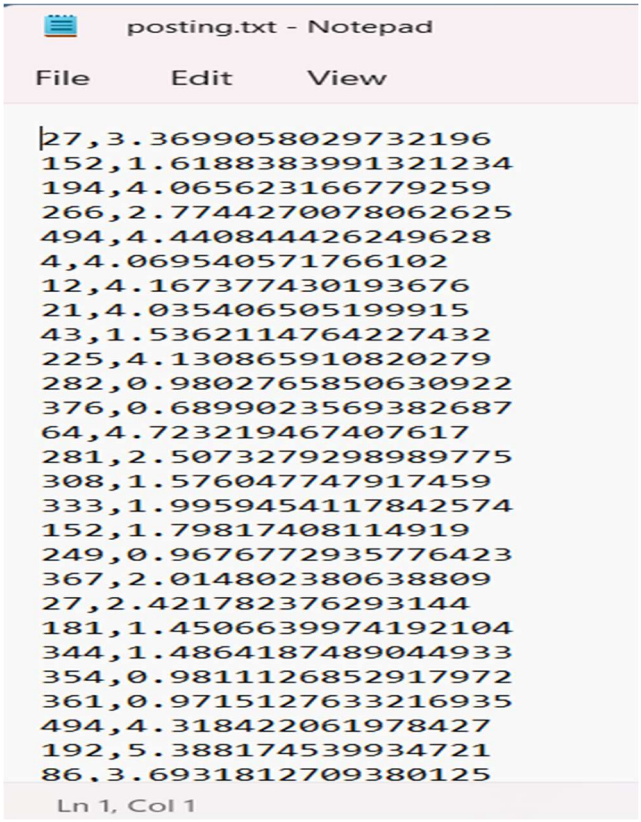
  "D:\IR\Assignment 1\newOutput"

Note: Place 'jsoup-1.14.3.jar' under the same repository. If not please provide the path to the jar while compiling and executing code.

Output:

A new directory output_files will be created at provided output path. Both "posting.txt" and "dictionary.txt" files will be placed under this newly-created directory. PFB screenshot as for both files. As we can see dictionary file is sorted based on terms.

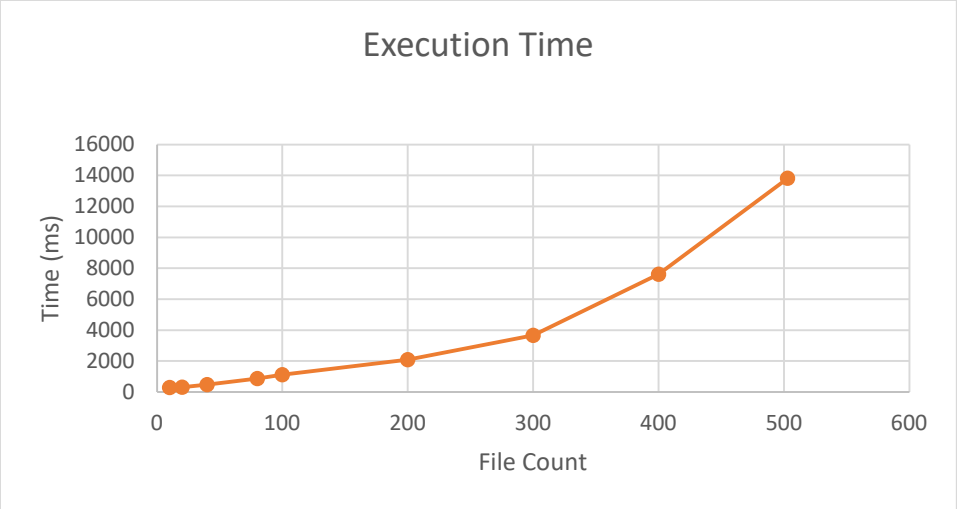


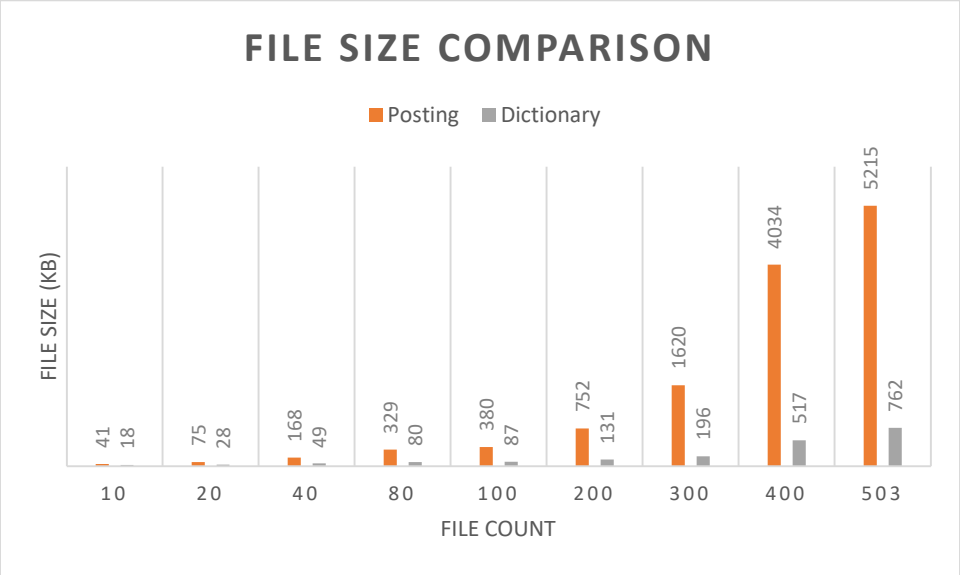


Performance & Memory Analysis

I have tested this code with a varying number of input files. Following observations were recorded and below are the graphs for the same.

File Count	Time (ms)	Posting.txt size (KB)	dictionary.txt size (KB)
10	298	41	18
20	310	75	28
40	470	168	49
80	863	329	80
100	1123	380	87
200	2093	752	131
300	3663	1620	196
400	7617	4034	517
503	13821	5215	762





Based on the above graphs we can see that both execution time and file sizes increase sharply as the corpus increases. Also, the growth of file size between 300 to 400 is much greater than the rest of the intervals. It can be inferred that these documents have a greater number of terms compared to other docs.

The average file size for the entire corpus is ~23.85kb(12MB/503) and that of documents between 300 to 400, it's ~35.7kb(3.57MB/100). This size is approximately 1.49 times higher than the average size for all of the corpus. This might be the reason for the growth observed in posting and dictionary file sizes.

Additional tests Performed

- Conditions handled two checks the correct number of input arguments
- This code fails to catch the text if the words are the result of some JavaScript functions.