Report of Phase Three

Steps to run the program:

You can run this program using the following command:

```
python simple.py <input_dir_path> <output_dir_path>
```

Please note that in order to run the program your working directory should be the project folder and you need to have beautifulsoup package installed.

Detailed summary of improvements made:

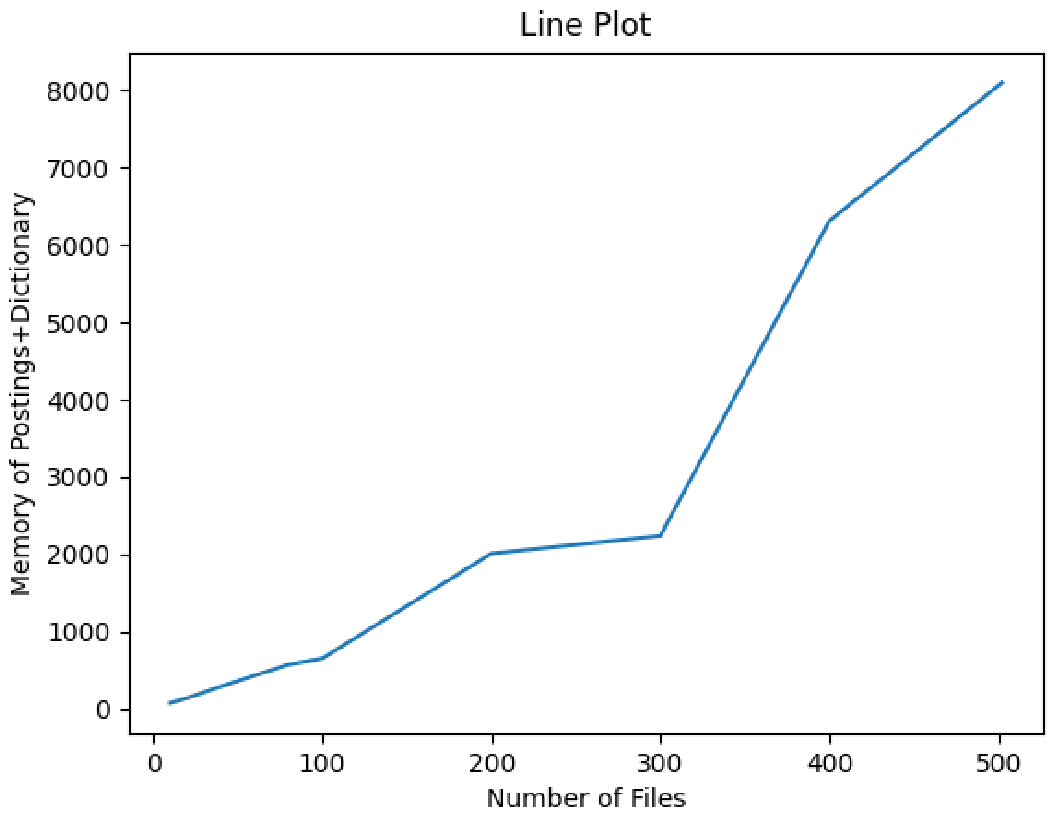
I have removed all the code which is related to creation of 502 output files which contains the tdidf values of each word that occurs in that file since it is not relevant to our project. I have also removed all the code that is related to BM25 scores since I used tfidf values in phase three.

For the tdmdictionary file, we are required to create a record for each word that has survived the preprocessing and for each word we should also give the number of documents in which it occurs and the position of that word in the postings file. I have created a global dictionary which contains all the words that have survived the preprocessing. I have created a dictionary named TDM which I used to maintain words and the number of documents it occurs in. I have taken each word from the global dictionary and looped through each word corpus of document to check if it exists in the document and finally created a record for that word in TDM with value as "number of documents it occurs in". I have also used another dictionary named postings_dict which I used to maintain all the words along with their postings file position. For each word in TDM, I have taken the "number of documents the previous word occurs in" and summed it by one to get the postings file position for current word. In this way, I have been able to populate all the values for all the words present in the global dictionary.

For the postings file, we are required to give the document name and the tfidf score of the word in that particular document. For this, I have created list named "global_tfidf_list" which will contain all the dictionaries of each document with the words in the documents and their tfidf values. I have taken each word in TDM and looped through all the documents to check if it exists in any of the documents and if it does exits in any of the document, we print the document name and the score of that particular word in that document. By using TDM as our reference for the words, we are able to maintain consistency between the postings file position we have given in the tdmdictionary file and the actual position in the postings file.

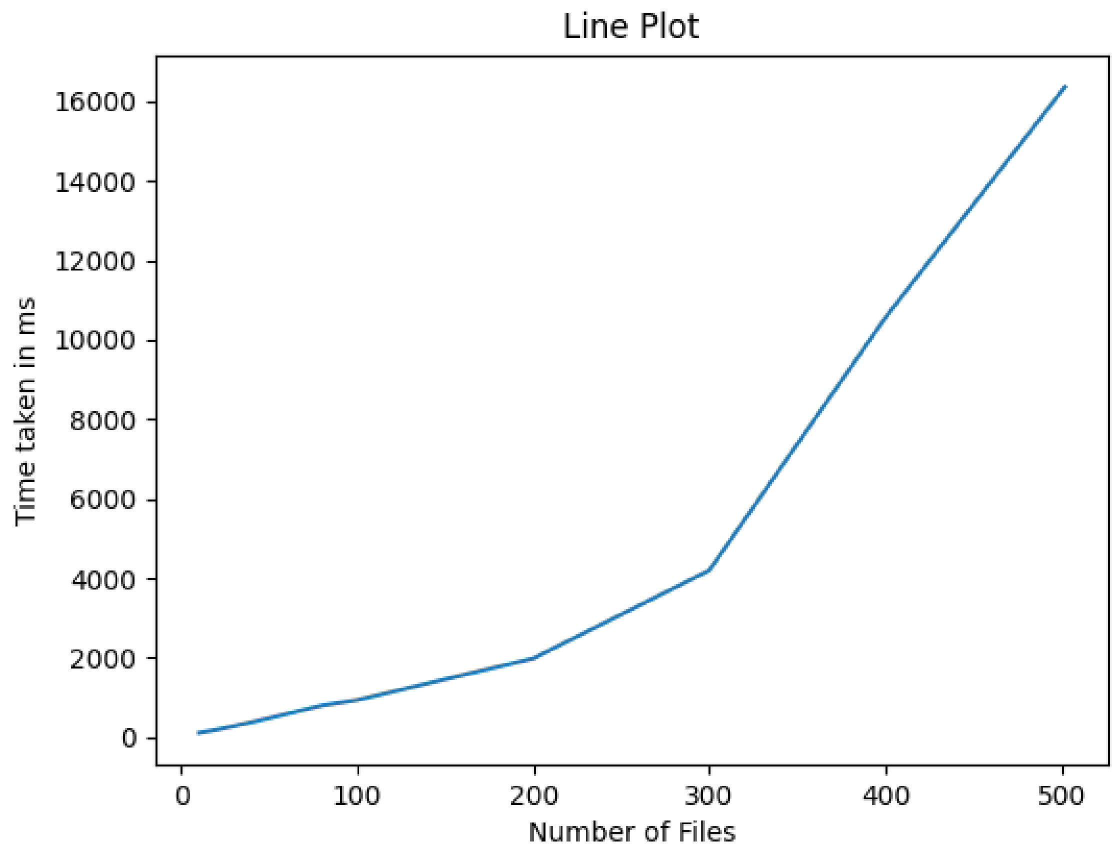
Comparing the size of raw data with the size of postings file and tdmdictionary file:

The size of the raw data before any of the pre processing is eleven megs and the size of the postings file and tdm dictionary file combined is eight megs. So, we are able to save almost thirty percent of the memory. The following is the graphical representation of the number of documents and the amount of memory postings file and tdmdictionary file consume together. From the graph you can see that there is an increment in the size of the files as the number of documents increase and there is a huge spike in the consumption of memory between 300 – 400 documents and if we were to analyze the sizes of raw data files from 300 – 400 they have much bigger size when compared to other files in the corpus. So, it is pretty evident that as the size of the corpus increases the size of the postings file increase.



Performance:

Number of Documents	Time in Milliseconds
10	115
20	190
40	380
80	799
100	938
200	1984
300	4200
400	10558
502	16364



Analysis: There is a slight increase in the time taken to preprocess all the documents and produce the postings and dictionary file. The reason for the slight dip in the performance might be the number of times we are iterating through each dictionary which we have each document. We are iterating through all these dictionaries to find the number the number of documents each word occurs in and we are also iterating through all the document dictionaries while writing the postings file in order to fetch the tfidf scores. This might have increased the time complexity which in turn affected our total time taken to run the program.