Safia Shah

CMSC 476/676: Information Retrieval

C. Pearce, Spring 2022

Phase 3: Building an index

1. **Abstract**

This report includes a summary and extensions made to the Phase 2 program in order to build an index for the input files given. The report will detail the changes made in order to create two outputs of a dictionary and the other a postings file. The records are not of fixed length as there is no need to assume the max token size.[[1]](#footnote-0) The postings file will contain a document id and the corresponding term weight for each surviving term. The dictionary file will hold the word, the number of documents that contain that word and the location of the first record for that word in the postings file. The program is a memory based implementation. The command to run the Phase 3 program is as follows:

**python3 index.py input-dir output-dir**

1. **Report**

For the preprocessing of Phase 3, there was not an extra requirement of filtration with the term words or stopwords. Phase 3 takes in the same command line arguments of input and output directories. I used the previous Phases 1 and 2 for the preprocessing as they seemed to be accurate in their approach. However, I had to remove functions that created the weight files for each input file. The term weights calculations were kept the same as in Phase 2 in which I used the tf\* idf[[2]](#footnote-1) score. As a reminder the term frequency was normalized by the document size.

In order to create the dictionary file, my approach was simple. Given the preprocessing and populating of the inverted index created in Phase 2, I would simply need to traverse through the term dictionary (TERM\_DICT). While traversing through the dictionary I appended the word/ token and used a previously made count function that returns the number of postings for a particular word. That gave me the token and the number of documents that contained that token. In order to find out where a token's first occurrence would be within the postings file, using a counter/index was sufficient. I created an index equal to 1 to account for the first word in the dictionary that would naturally be the first posting in the postings file. In order to find the locations in the postings file for each token after I added the previous token’s document count to the index and it gave me what line the next token’s first occurrence would be. The dictionary is in hash order determined by the TERM\_DICT within the program. The following is an example of this process on the TDM from the project documentation :

The Postings File (PF) Dictionary Work

1,0.2 cat PF = 1(start)

3,0.3 2 docFreq = 2, index = 1

2,0.1 1 (PF line)

3,0.1 dog

1,0.1 2

3,0.4 3 (PF line) PF = index + docFreq = 3

Rat new index = 3, new docFreq = 2(dog’s posting]

2

5 (PF line) PF = index + docFreq = 5

This pattern was repeated for all tokens within Phase 3 and a quick test of printing the word in the postings list also showed the dictionary calculation matched up with the postings file output.

The postings file was created by simply iterating through the term dictionary and then for each bucket traversing through each posting. The postings node contains the document id and the term weight calculation and all postings are in document id order as a default. All I needed to do was write each posting's doc id and weight into the postings file by editing a previously made linked list function and passing in all needed parameters like the output file name. Both the postings file and dictionary file creation was done in the writeOutput() called near the end of the index.py file.

1. Efficiency

The efficiency of the project proved to be much faster than the previous two phases. Phase two created all 503 file outputs with the token and their weights. The jump in Phase 2 timing also increased significantly with the processing of 300 and 400 documents. This is possibly due to the tokenization of html files that are in different languages and/or that may be in different formatting than a basic html text file. Some files may require more removals of stopwords and preprocessing like it was done within Phase 2. The execution time is still significantly less than Phase two because there is not a maximum of 503 output files and therefore no frequent opening and closing of outputs when writing the tokens and their weights. In Phase 3 there is an increase of about 6 seconds between the 300-400 documents and 400-500 documents which makes me assume the time will start to increase by 6 seconds for every 100 documents past 400 if those documents are of the same complexity. On this smaller scale the timing is not bad, but if the jump in execution time for every 100 documents increases as shown below it seems like it would not be efficient for a very large corpus.

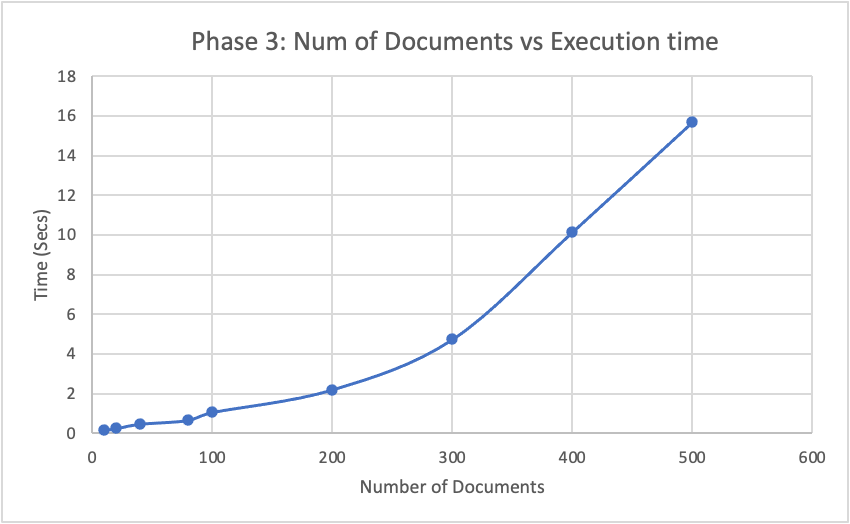


Figure 1: Time vs Num of Documents Graph

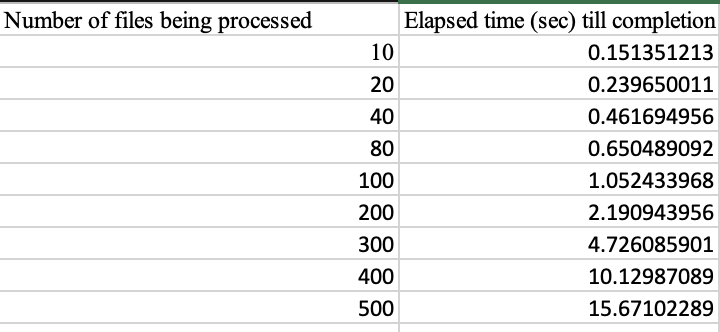
**

Figure 2: Time vs Num of Documents Plot Points

1. Total size of Output files ( dictionary and postings) cmp Size of input files

[ Discuss how the total size of the output files, dictionary+postings, compares to the size of the input files, for various numbers of input files. ]

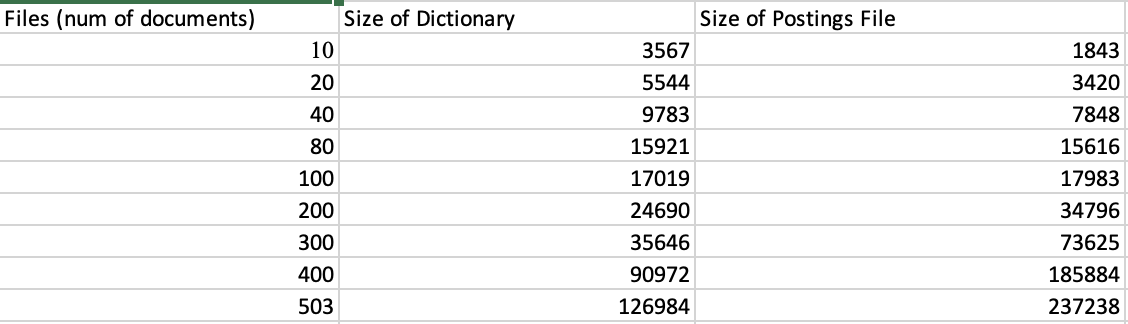


Figure 3: The size refers to the number of lines in the documents.

Above is a chart of the number of lines created for each set/ number of documents. If I were to compare a single input to its respective single output file for a dictionary and postings file, I assume the outputs would be smaller. This is because the input document goes through preprocessing that strips a lot of the punctuation, words that occur once, words of length one, stopwords, and the html that is not needed. The postings file has a line for each surviving token and the dictionary is in theory, and for a lot of the inputs, smaller than the postings because for each token there is a max of 3 lines to give information for it.

It is also interestings to compare the size of the input files and the output files on disk. For the total of 503 documents the output files together come to the size of 4.3 MB (dictionary - 789kb, postings 789 KB) uncompressed while the size of the input files comes out to 12.7 MB uncompressed. This is consistent to my observation above of the output files being a lot less in size compared to the inputs. For a set of the first 10 inputs the total size comes out to 129kb while the output files for the first 10 comes to about 24kb for postings and 18 kb for the dictionary. This trend continues for every other set number of documents. The results of this Phase show that it is more efficient to use an inverted index and one of the best ways to store it is using a dictionary and postings file that correspond to each other rather than storing each token in its respective file separately. One is able to see correlation, weight of a token, and the instance of a token and use that to compare various different input files.

1. “Don't worry about the variable word length” - Dr. P [↑](#footnote-ref-0)
2. *tf\*idf = [freq of word w in doc i / total terms in doc i] \* [num docs in corpus / num docs that contain word w]* [↑](#footnote-ref-1)