Information Retrieval Programming project -3 Report

Introduction:

The main aim of this project phase two is to create a dictionary file containing the term , doc id and to locate the first document that contains the term. There is one more output file which is the postings file. The postings file contains the term and its normalized weight. I have used python as programming language for the code. Windows Operating system 10 was utilized for the project. The following is the command to execute the code .

$python calcwts.py files output

where,

index.py – python code

files – input directory

output-output directory

Input :

The inputs for the project are the 503 input html files which has to go some preprocessing to extract only the content leaving the html tags from those 503 files. The content of the input files are tokenized and given as input for the term weight and to create the index for the terms. The input files are from 001 to 503 html files with .html extension.

Output:

The output files are the .wts files which contains the dictionary file and the postings file. As mentioned above, dictionary file contains the term , doc id and first document that contains the term. There is one more output file which is the postings file. The posting file contains the term and its normalized weight

Implementation work:

The input files containing the html files are used to extract only the content of those pages by importing and utilizing the html2text python module. This extracted content is tokenized using nltk tokenizer available in python. In this programming project phase two , I have removed the list of stop words from the tokens by iterating it through the list of stop words in a separate file stoplist.txt. This improves the quality of tokens in the documents . The stop words removed tokens in the documents are stored in a nested dictionary containing files names as keys , values are tokens and frequency. This dictionary is iterated and the length , average length and total length of the documents are calculated. I have implemented two separate classes one for tokenizing the documents and one for calculating the term so that the program is modularized in a better way. I have also created a separate class for index that have a nested dictionary to store the term, document id and the first location of the word in the document The program is design using object oriented principles. I have created the index using python nested dictionary that has keys term , doc id and value as the first location of term in the document.

Indexing:

The index is built using the nested dictionary in python . For each term, the id of document which it appears in as well as normalized weight of the term is written to the output postings file. A counter variable was used to record the last line number written in file. After going through all documents corresponding to a term, a single entry for that term is stored in a list as tuple (of the form term, document frequency, location of first record in postings file). The document frequency for a term was obtained from the inverse document dictionary (inverse_doc_freq[term] returns number of documents that contain that term) built in phase two. After going through all the tokens, the list of tuples was sorted according in alphabetical order of terms and the result was written to the dictionary file

Term weights formula:

The term weight is calculated by the taking a product of term frequency and inverse document frequency.

Term frequency = Frequency of a term in a document / total frequency of the words in all the documents

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{i,j}}$$

The above is the formula for the calculating the term frequency in the documents where

$n_{i,j}$-number of terms i in a document of j

$\sum_k n_{i,j}$ - Total frequency of words in all documents

$tf_{i,j}$ - Term frequency

Inverse Document Frequency :

Inverse Document Frequency is the logarithmic division of n documents in the whole collection by the ni which is the number of documents in which the term occurs . Idf is computed for the entire corpus whereas tf is computed on a document by document basis.

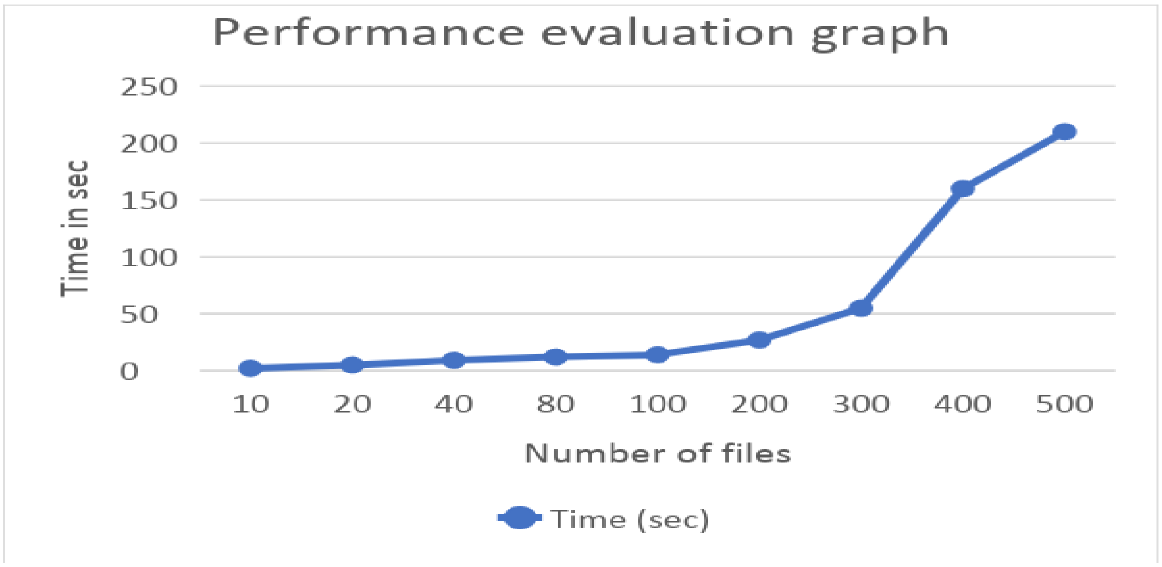
$$idf_j = \log\left[\frac{n}{df_j}\right]$$

The Term weighting product is calculated by taking the product of term frequency and the idf. The formula is shown below .

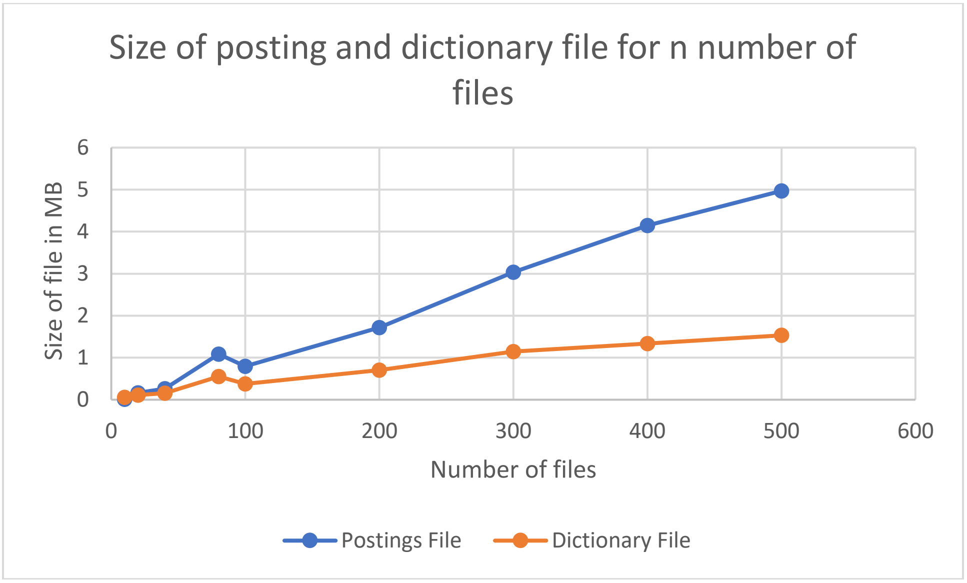
$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

Performance Evaluation:

The performance of the code was evaluated by considering the time between the start time and end time. I have considered the time taken by 10 , 20 , 40, 80, 100, 200, 300 , 400 , 500 files . It is clearly seen that the graph grows linearly for the addition of the performance of the files from 001 to 500 files in the output directory .The x axis denotes the number of files and the y axis denotes the execution time in seconds . The graph for the performance evaluation is shown below

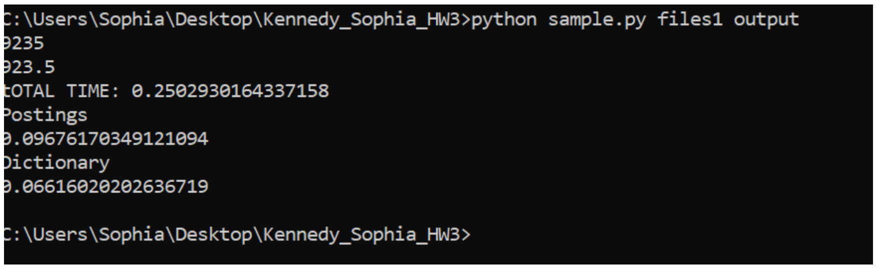


I have also calculated the size of the posting and dictionary file for the number of given input files.



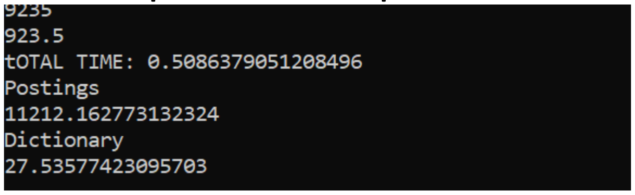
Result:

The below is the output for the given 503 input files.There are two ouput files one is dictionary and the other is postings.



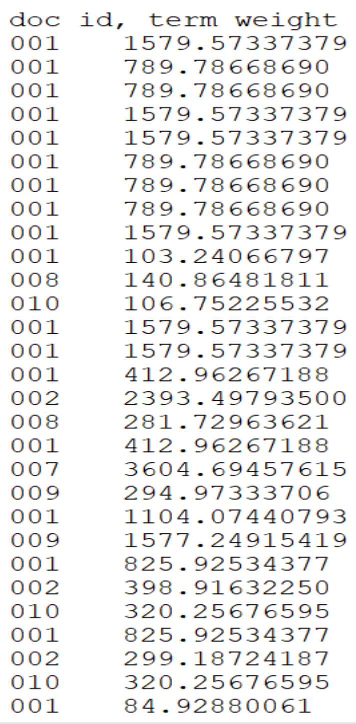
```
:\Users\Sophia\Desktop\Kennedy_Sophia_HW3>python sample.py files1 output
235
23.5
OTAL TIME: 0.2502930164337158
ostings
.09676170349121094
ictionary
.06616020202636719

:\Users\Sophia\Desktop\Kennedy_Sophia_HW3>
```

The above picture is the Output



```
9235
923.5
tOTAL TIME: 0.5086379051208496
Postings
11212.162773132324
Dictionary
27.53577423095703
```

The above picture is the output for the 10 files



```
doc id, term weight
001    1579.57337379
001    789.78668690
001    789.78668690
001    1579.57337379
001    1579.57337379
001    789.78668690
001    789.78668690
001    789.78668690
001    1579.57337379
001    103.24066797
008    140.86481811
010    106.75225532
001    1579.57337379
001    1579.57337379
001    412.96267188
002    2393.49793500
008    281.72963621
001    412.96267188
007    3604.69457615
009    294.97333706
001    1104.07440793
009    1577.24915419
001    825.92534377
002    398.91632250
010    320.25676595
001    825.92534377
002    299.18724187
010    320.25676595
001    84.92880061
```

The above picture is the output for posting file.

Conclusion :

We can see clearly from the graph that the time grows linearly and sub-linearly based on the increase in number of files. Sorting of files are done using merge sort .Hence the total time that must have been taken would be O(nlogn).