**SHARAD DIXIT**

**CMSC 676 INFORMATION RETRIEVAL**

1. **OBJECTIVE**

The objective of this assignment was to build an inverted index for the documents of the collection. The project developed by me takes each file one by one and then parse each token and store it in the inverted index. The stop words and single letter words are also removed.

1. **ARCHITECTURE**

Step1: HTML Parser to receive tokens

Step2: Stop words and single character word removal

Step3: Each Token getting extracted then storing in Hashmap as Key and Value as an ArrayList

Step4: The ArrayList which is the value of Hashmap has ArrayList as the value that is value is ArrayList of ArrayList. The inside ArrayList consist of double values as file number from which the token is coming and as well the tf\*idf weight of the token.

Step5: Output file is written for each token, where the dictionary file consists of the following items:

1. Token
2. Total number of documents that have that token
3. Count line number (From the posting list file)

Step6: The other output file (Posting file) generated is written with each token file number with corresponding weight of the token. This is done for each token and the weight and file number wherever it is present in the corpus is written.

1. **APPROACH**

The project for developing the inverted index proceed further by using the output obtained from assignment 2 that is weight of each token. The following steps were taken to develop the project:

1. Hashmap created with key as a string(token) and value as ArrayList (ArrayList of ArrayList)

The inside ArrayList has two double values as File Number and Weight of Token.

1. The tokens are generated one by one as each file is parsed and then tf\*idf weight is calculated.
2. After the calculation of weights, the tokens are stored in the Hashmap and side by the value of File Number from which the token has come is stored in the ArrayList and the weight calculated is stored as the second value in array.
3. After each file is parsed all the key and value are stored in the HashMap.
4. The HashMap key values are further sorted in alphabetically order so as to find the correct location in the ‘Posting file’
5. The first output file is generated where the ‘Dictionary file’ consists of the token, total number of documents that consist that particular token and also count line number from the posting list.
6. The second output file consist of each token’s file number and then corresponding weight of that token. For each token all the Files which the token is presented is given as the output with the corresponding output.

Since, the HashMap all the tokens were alphabetically order therefore the count of the ‘dictionary file’ helps in getting the location in the ‘Posting file’

1. **EVALUTAION OF PROGRAM**

* The output files obtained that is the ‘Dictionary File + Posting File’ size in total is (3.4MB + 10.8MB) 14.2MB which is in comparison to the input file larger which is about 12.7MB for the whole corpus. This is shows that the dictionary file and the posting file obtained has all the data needed to cover the input files and as well as these files can now help us to point out that which token is present where and in total of how many documents.

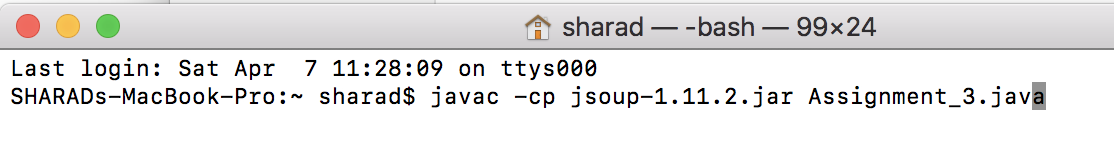
For beginning 100 documents the ‘Dictionary File + Posting File’ size is in total (360KB + 800KB) which is comparison to first 100 files that is near about (1100kb) still larger and hence still the satisfies the above point of all data being covered and as well as pointing towards that which token is from which file.

* The time efficiency of my project is similar to that of Assignment 2 as still all the processing for development of tf\*idf weight is done but the only difference is that the index is developed when the tokens are getting generated. This can help us as in a real query search engine as pre-processed weights of all the tokens is not needed which is done presently as because only those words which come in the query are needed to be weighted so that the ranking can be done, and those documents can be retrieved.

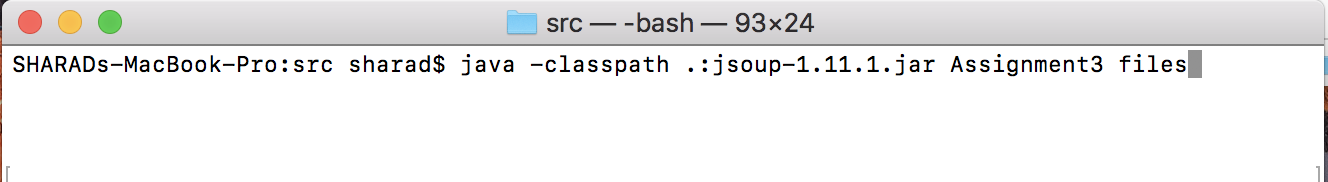
1. **OUTPUT**

Below is the script to test the project from my environment.

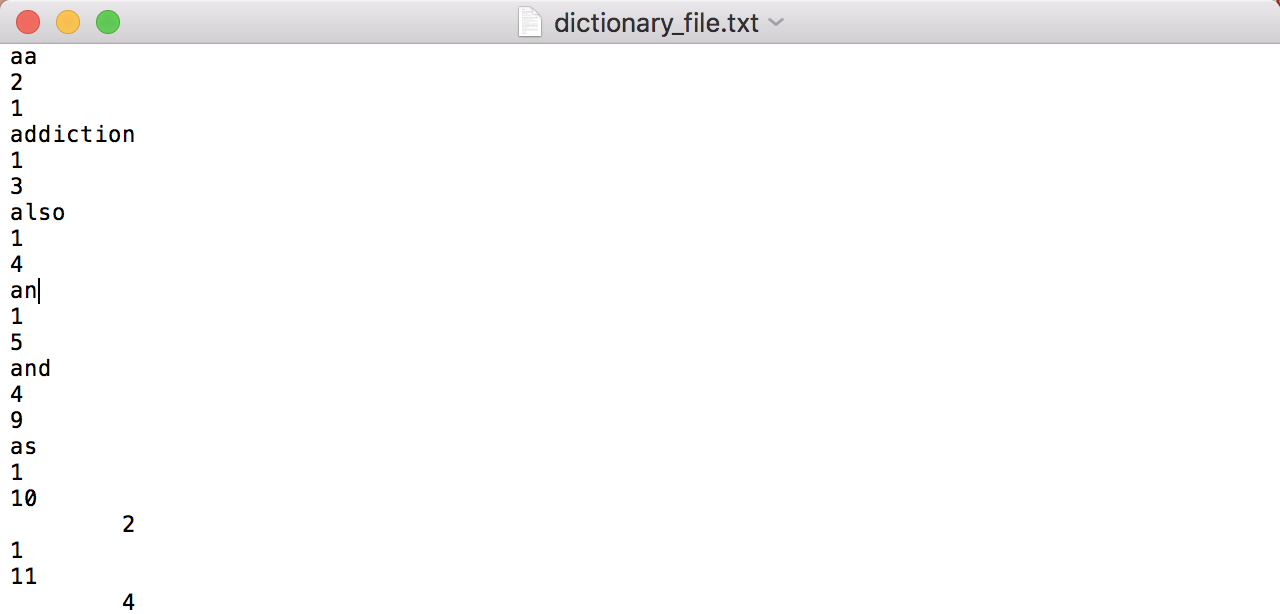
1. For compiling the source code in order to obtain the java class file.



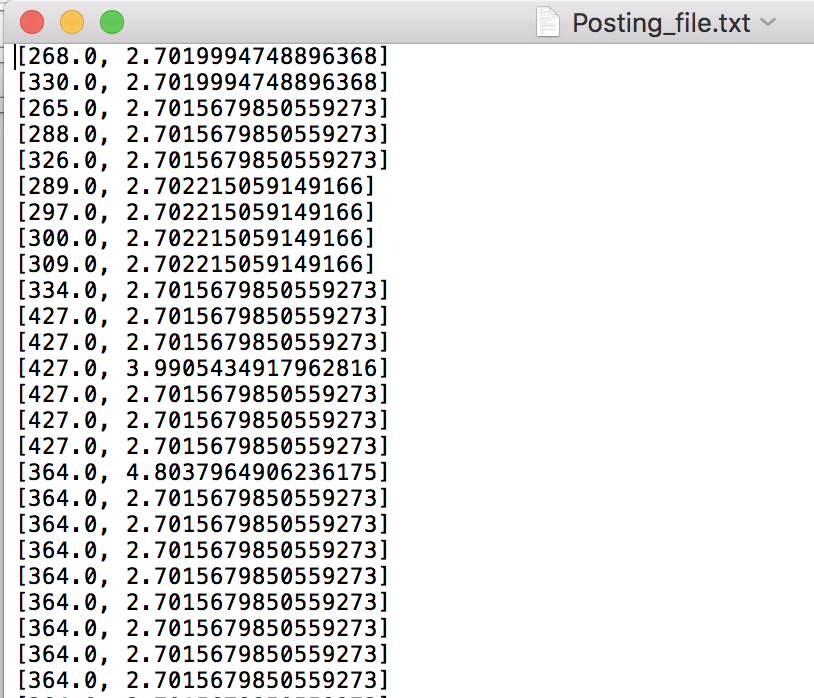
1. For running the obtained java class file to get the output file (input file: file path of directory)



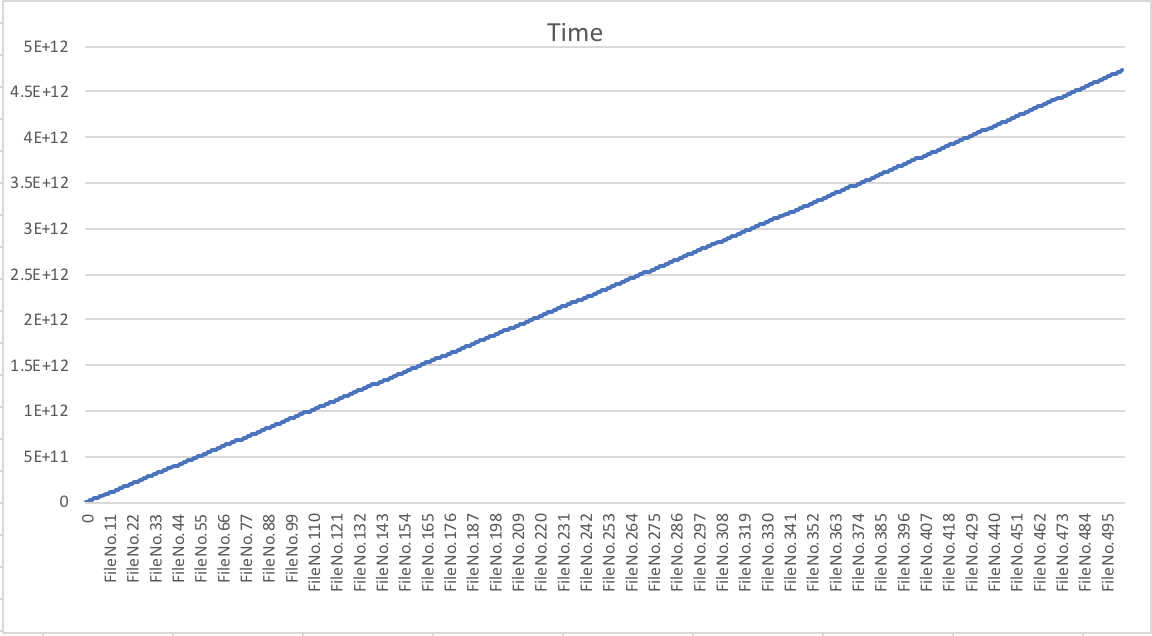
1. Dictionary File:



1. Posting File:



1. Graph (Indexing Time v/s Number of Documents)



1. **Limitation with Reason**

The limitation to my approach is still the same which was in Assignment 2 that is for each document when parsed and tokens are collected then all the tokens are stored in Hashmap and for each document a Hashmap is created which is expensive task. Since, the tokens are stored in a Hashmap instead of an ArrayList therefore the computation time is less, but the task is more expensive.

**7.) Working of Program**

**A) Pre-Requisite**

# Input the whole directory as the input for my project, that is just the filepath for the input directory is needed in order run the project.

**B) Developing Environment**

# Mac OS with 16GB RAM

**C) Submission Package**

# Report

# Zip of Output files

# Directory with: Assignment\_3.java (source code)

: Assignment\_3.class (for running of file)

: stopwords.txt

: jsoup-1.11.2.jar (HTML parser library)

: test files

: OUTPUT FILES: 1.) dictionary\_file.txt 2.) Posting\_file.txt