**SHARAD DIXIT**

**CMSC 676 INFORMATION RETRIEVAL**

**Assignment 2**

**1.) OBJECTIVE**

The objective of this assignment is to calculate the term weights for each token in each document. The project developed by me completes the requirements of calculating weights of each token by the use of tf\*idf variants and then further normalizes the weights by using weighted scheme. The project also successfully removes the stop words and then calculate the weights of the tokens.

**2.) ARCHITECTURE**

Step 1: HTML parser

Step 2: Tokens extraction on space basis

Step 3: Removal of unwanted tokens

Step 4: Removal of stop words from tokens

Step 5: Storing Each file tokens in particular Hashmap

Step 6: Calculating term frequency

Step 7: Calculating inverse document frequency

Step 8: Calculating tf\*idf

Step 9: Normalization of weights

Step 10: Writing a file with tokens and weights

**3.) APPROACH**

The tokenizer developed in project has been used enhanced as it now handles the terms which include e-mail address, IP address and also handles other condition. The limitation that were present in the project 1 has also been overcome like computation time and also treats terms like “wood-of” as a single term which were removed earlier in the project and hence the tokenizer has been enhanced. Step1 to Step 3 were a part of the Project 1.

For the implementation of project 2 following approach has been taken:

**Step3**: Removal of unwanted tokens which were basically garbage value in some of the files were removed so that those does not decrease the token weights and decrease the proficiency for ranking.

**Step4**: Removal of stop words from the tokens obtained, as these words do not provide any information of the document and hence are not needed for ranking of a document. This is done by removing stop words from each document when the document is parsed, and tokens are obtained from the document.

**Step5:** After removal of stopwords then each file tokens are stored in each Hashmap, that is a file is assigned with a designated Hashmap and all the Hashmaps are stored in a list of arrays so that they can be accessed later. This methodology however is expensive but is fast as compared to storing in an array list and then linearly traversing in the array.

**Step6**: After removal of stop words then term frequency of each term in each document is calculated. Term frequency is defined as the number of times a term occurs in a document that is basically count of a term in a document. The method used for to calculate term frequency in the project is tf(word)=1+Math.*log10*(tfword), where tfword is the term frequency of term in a document. The term frequency is calculated because a document with a term with more occurrences is more relevant than a document with the same term with less occurrences. The above stated is true but not linearly and hence we need some scaling mechanism for calculating the document score with term frequency. Therefore, the term frequency method provides us with good scheme for calculating a document score.

**Step7**: After calculation of term frequency inverse document frequency is calculated. Inverse document frequency is defined as in how many documents a term occurs. This scheme is calculated in order to see that in how many documents a term occur. The method used for to calculate inverse document frequency in the project is idf(word)=Math.*log10*(totalfiles)/(idfword), where idfword is the inverse document frequency of a term in all the documents. The inverse document frequency is calculated as rare terms are more informative than frequent terms in a document like for example stop words like ‘the’, ‘a’ have are very less informative when calculating page rank and score therefore are eliminated.

**Step8**: After calculating term frequency and inverse document frequency then tf\*idf is calculated for each term in each document. This scheme is the best weight scheme and the method to calculate tf\*idf= tf multiplication idf. The tf\*idf increases with the number of times a term occurs in a document and increases with rarity of term.

**Step9**: Normalization of each term, this is done by using the method

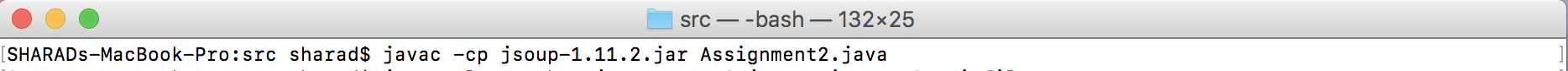
W(i) = W(i)/sqrt(W12+W22+W32+….+). This normalization considers document length into consideration.

**Step10**: This step writes each term with its normalized weight for each document and creates a new file for each document.

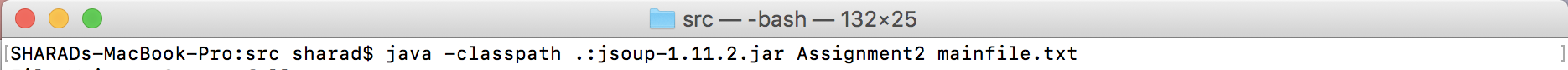
**4.) Output**

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

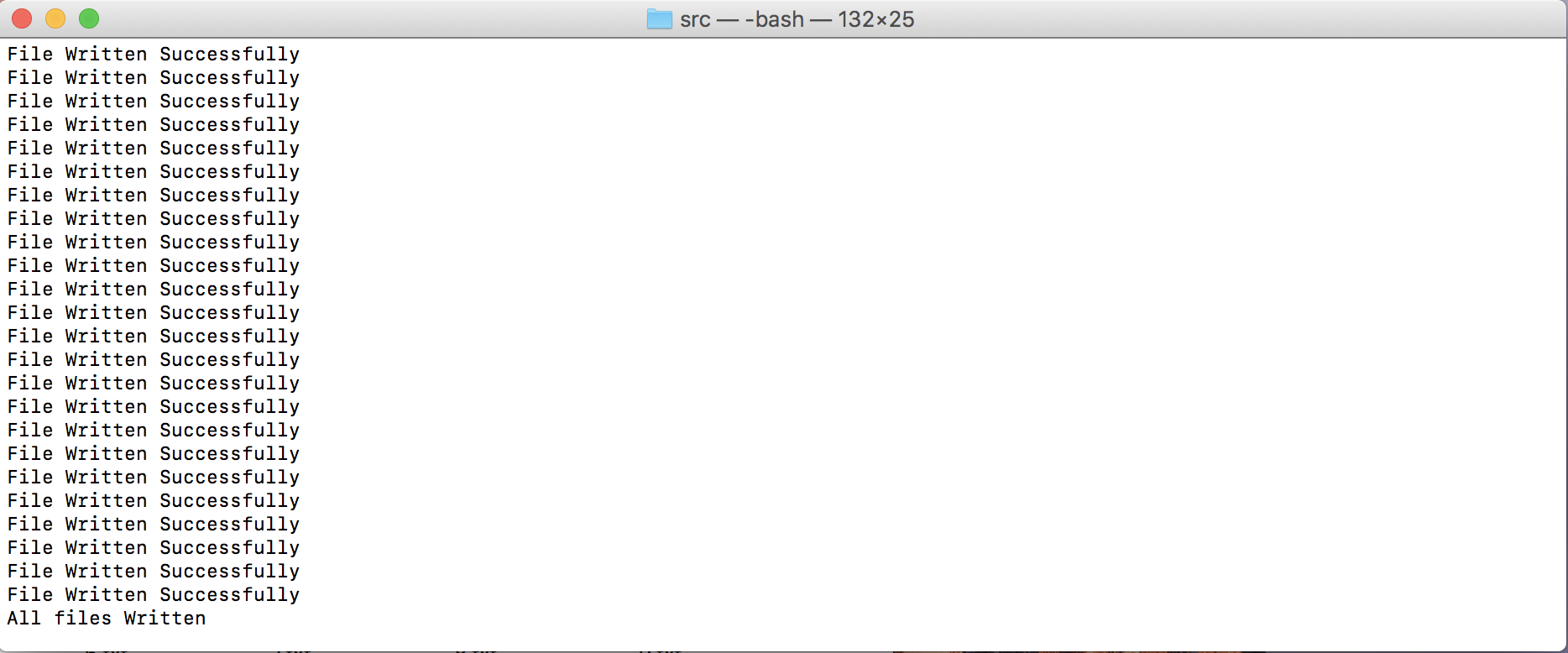
1. For compiling the source code in order to obtain the java class file

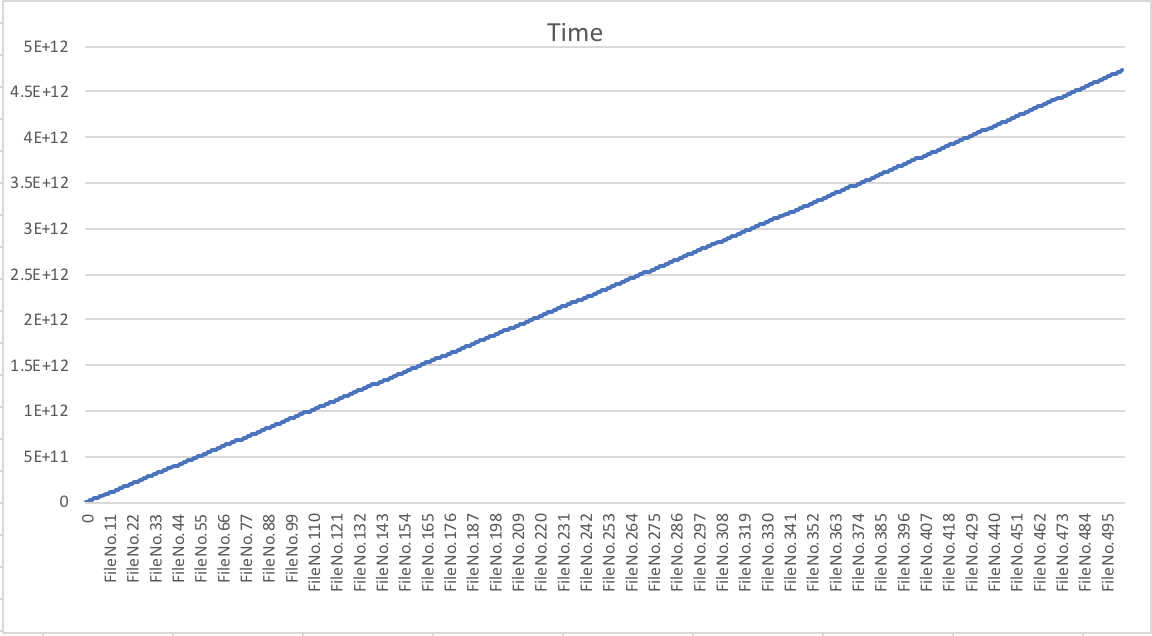


1. For running the obtained java class file to get the output (input file: mainfile.txt)

****

**OUTPUT Of CONSOLE**

****

**5.) Graph(Number of Files v/s Time)**

The above graph shows that with increase in the number of files, the time complexity of the program also increases.

**6.) Limitation with Reason**

The only limitation of my method is that 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. As for calculating idf the all the documents are needed to be linearly searched once therefore, I had two choices either each documents tokens stored in an array or in a Hashmap so that then linear search could be done. When implementing the task with an array list then the efficiency was less as each array list needed to be traversed in a loop whereas with Hashmap linear search was needed as it points to the key values. Therefore my method has less computation time but expensive.

**7.) Working of Program**

A) Pre-Requisite

# Input the file with the name of each file to get the output is created which is named as “mainfile.txt”

# In order to obtain the output the mainfile.txt is given in the command line and the output is obtained in the same folder (with the class).

B) Developing Environment

# Mac OS with 16GB RAM

**C) Submission Package**

# Report

# Zip of Output files

# Directory with: Assignment2.java (source code)

: Assignment2.class (for running of file)

: mainfile.txt (input file for 500 files)

: stopwords.txt

: jsoup-1.11.2.jar (HTML parser library)

: test files

: Output\_files.zip