CMSC 476/676 - Information Retrieval
Homework 2 - work individually
Due: Monday, March 11, 2019 @ 11:59 pm

The objective of this assignment is to calculate the term weights for the tokens that occur in each document in your collection. To execute your program, it should be sufficient to type:

```
calcwts input-directory output-directory
```

where the input-directory contains the original files (i.e., the unprocessed `files` directory) and the output-directory contains all generated output files. The calcwts command may be a lex/C/C++/Java program or a shell script that calls a series of programs written in C/C++/Java/UnixTools and/or lex/Java/C/C++/Perl/Python/Php/etc. It is permissible to create *at most* one set of temporary files. For example, you may tokenize and count frequencies within documents using Perl or php and output that information to one temporary file per input file.

Preprocessing You may choose to build on either preprocessor created in your Homework 1. If you are not satisfied with your preprocessors, you may choose to borrow a preprocessor from a classmate (but attribute the source). You must **extend** the preprocessing as follows: remove stopwords, remove words that occur only once in the entire corpus, and words of length 1. Use this list of stopwords.

Term Weighting You may use any of the tf * idf variants discussed in class and/or the textbook to weight your terms. Be sure to clearly describe the formula you chose to use in your report. You must normalize the term weights by the length of the documents. The easiest normalization is to use freq(wordi in documentj) / totalfreq(all tokens in documentj), but you may choose to do the proper sqrt of sum of squares vector normalization or any other normalization that takes some measure of document length into account.

Output Files The goal is to build one output file per input file. In the output file, there would be one line per token that survives preprocessing. That line would contain the token and the token weight.

Algorithm You may implement either a memory-based or file-based algorithm, i.e., you may create a hash table in memory for each document to store the information about term frequencies, or a temporary file on disk to store the term frequencies (deprecated). You will store the global information (numdocuments per term) in memory. You should time your term weight calculation program on a varying number of documents (10, 20, 40, 80, 100, 200, 300, 400, 500) and plot a graph of the indexing time as a function of the number of documents in your report. The timings should be done on the FULL process (starting with the raw documents).

Program Testing
Use the same set of files as in Homework 1.

Program Documentation.
After your internal documentation (comments) are complete, write a report which provides an approximately three page summary of your program. In particular, discuss how you extended/improved the preprocessing from Homework 1, if you made any changes. Show two examples of input documents and the resulting (surviving) tokens and their weights. Discuss the efficiency of your approach in terms of your timings on a linux machine (use the time command) as a function of number of documents indexed. The entire document should be four pages maximum. I will primarily be grading from the report, so make sure it clearly describes what you did and what your program's output and efficiency.

Hand In
A zipfile or similar, including shell scripts, the report in pdf format, and using 025.html, give me 025.wts, i.e. the term weighted document. Please name the zip file in the form LastnameFirstnameHW2.zip . Email the zip file to the course's TA.

Late Policy
10% deduction per 24 hours. Assignments turned in during or after the class will result in a 10% deduction.

Grading Rubric
Program runs to completion (50 points)
Performance analysis (table or graph, for different numbers of input documents) (15 points)
Report, include discussion of term weighting scheme used, (35 points)
and some details TBD

Extra credit for implementing BM25 (term weighting), using suggested values for K1 and S1, or explain your own choice

Extra extra credit for implementing this using a character n-gram approach (without stopping or stemming.)

Extra extra extra credit for something clever on your part, such as a recursive hash function in your inverted index ... :-)