CS433-CS533 Project Part 2
(Due: March 24, 2020)

The goal of this project is to develop a small IR system and to practice implementing different components of an IR system. This project will be carried out in multiple parts during this semester. This is the Part 2 of this project.

The goal of this part is to build a basic query processing component for the IR system based on the dictionary and the postings lists that were constructed in Part 1 of the project. So the input to this part of the project is dictionary.txt and postings.txt. Since some students did not generate the correct dictionary.txt and postings.txt, we recommend that all students use the dictionary.txt and postings.txt provided by the instructor for Part 2 of the project.

Again, the project is a team project with two members of your own choice (one team has three members because we have an odd number of students).

Your project for this part needs to accomplish the following tasks:

1. Build a fast access data structure for searching terms in the dictionary. Your data structure can be either a hash table (simplest) or a B+tree. If you choose to implement a B+tree, you can implement a simpler static (assuming that no document will be added/deleted/updated) and memory-based version (i.e., the nodes do not need to be a physical page). In your implementation, each internal node and leaf node should accommodate up to 10 entries, where an entry is a (term, pointer) pair for internal nodes and a (term, df, pointer) triplet for leaf nodes.
2. Convert each term frequency value in the postings lists to a tf weight value using formula $1 + \log_{10}tf_{t,d}$.
3. Compute the document length normalization factor (1/||D||) for each document and store all the normalization factors in an array. Only tf weights are used to compute the normalization factor.
4. Parse input queries. Vector space (free text) queries need to be supported. For each query word, it needs to go through the same tokenization process you used to parse document terms in Part 1 of the project to generate a query term.
5. Find the correct postings list for any query term. This is done in three steps. First, find the correct dictionary entry for any given query term using your fast access data structure. Second, from the dictionary entry found, retrieve the location of the postings list for the query term. Third, retrieve the postings list.
6. Implement the Document-At-A-Time algorithm for processing vector space queries based on the Cosine similarity function.

   When a query comes, for each query term, use $1 + \log_{10}tf_{t,d}$ to compute its tf weight and use $\log_{10}(N/df_t)$ to compute its idf weight (the df value comes from the dictionary entry for the query term). The overall weight of the term is the product of its tf weight and its idf weight. When computing the similarity for each document, do not normalize using the query length (not needed as it does not affect ranking). For each query, sort the results (retrieved documents) in descending order of the similarities. For each result, output its docID and the similarity value.
7. Show the top 10 results for the following 4 queries: (1) europe; (2) stock rally; (3) debt crisis; (4) stock future higher. If a query yields less than 10 results, just show all the

results. The results (i.e., (docID, similarity) pairs in descending similarity value) for each query should be saved into a text file. The file name for the first query result should be query1result.txt. Similar for other query results.

Programming language requirement

You may write the code using any programming language but it is required that your program must compile on harveyv.binghamton.edu. No exceptions. It should be purely a command line program. NO GUI will be accepted. This enables the TAs to run your code using test scripts. DO NOT assume that since your program compiled and ran correctly on your laptop it will also compile and run correctly on harveyv.binghamton.edu

What to submit?

You need to send a [file_name].*tar.gz* file to the TA by the due time. The file name should contain the last names of the team members. When the file is unzipped it should contain a directory with the same name as the zip file. The directory should contain the following files:

1. A page with the following and signed by all team members: “We have done this assignment completely on our own except for the software/tools acknowledged in the project report. We have not copied it, nor have we given our solution to anyone else. We understand that if we are involved in plagiarism or cheating we will have to sign an official form that we have cheated and that this form will be stored in our official university record. We also understand that we will receive a grade of 0 for the involved assignment and our grades will be reduced by one level (e.g., from A to A- or from B+ to B) for the first offense, and that we will receive a grade of “F” for the course for any additional offense of any kind.”
2. The source code of your implementation plus possibly a make file. The code should be reasonably commented.
3. A readme.txt file on how to run your code.
4. A status report of your project. It should report what programming language is used, how complete your implementation is (especially for students who could not fully complete the project), whether correct results are obtained and the number of terms in your dictionary.
5. The revised postings.txt. After Project Part 1, the postings are pairs of the format (docID, tf). In the revised postings.txt, the postings are pairs of the format (docID, tfw), where tfw is computed using formula $1 + \log_{10}\text{tf}_{t,d}$.
6. The result files query1result.txt, query2result.txt, query3result.txt and query4result.txt.

Where to submit?

Submit to the Project Part 2 submission folder in MyCourses.

Demonstration

The instructor/TA may ask teams to demo their projects as needed.

Plagiarism Check

All your code will be subject to check for similarity with other submissions using Moss. So you are advised not to look at each other's code.