Phillip Thompkins
CMSC476 Spring 2014

HW4 Writeup
Command Line Retrieval Engine

Prior Versioning and Pre-Processing

Even though I received a good grade on the preceding assignment, I doubted the validity of my results from Homework 3. So, to be safe, I used the Dictionary and Postings files of classmate Sharyn Kurland for my command line retrieval engine. Since our files only differed in the numbers and not in the formatting, my code is mainly unchanged from what it would have been had I used my own results files. And since we didn't have to do any direct processing with the documents themselves now that we had the Postings and Dictionary files (thanks again to Sharyn for that tip!), I gutted my code from Homework 3, and essentially started from scratch, focusing solely on interfacing with the Postings and Dictionary documents.

Implementation

I feel that my implementation was rather simple. The queries from the command line were placed into a list for easy access and traversal. The command line retrieval mechanism, referred to in-program as wordCorpus, was created by use of a generic python hash table, with the term itself as the key and value was a list, consisting of pairs of document-names and weightings within that document.

With the wordCorpus populated, the program then moves to search for each term in the list of queries. For each term that is found, the results within the wordCorpus are added to a list that records relevant results. If a term is queried in a document more than once (if someone searches, for example, "dog eating hot dog" on the command line) then the pre-existing search results become additive. That is, results for "dog" in document #4 would be greater than the results for "eating" or "hot".

After the program has gathered its list of relevant results, it must go through and sort the results. I was informed that in class, we were told to multiply the amount of times a term occurred by its frequency, to better account for queries like "dog eating hot dog". So, that happens first, and nothing of interest happens if the term only shows up once in the query. After that, the results are sorted, most favorable results first, and printed in order. No more than ten results are printed.

Algorithms

As a disclaimer, I have yet to take the Algorithms course, and it has been quite a while since I have taken Data Structures, so my analyses may be a bit off.

I have broken my analysis into the following three sections:

- wordCorpus population: There are two loops in this section, one that runs a number of times equal to one-third the length of the Dictionary file, and another that runs once for each file a term shows up in. If we assign “dictionary file length” to the variable *d* and number of files a term occurs in to *f*, then the complexity of populating the wordCorpus can be expected to be roughly *(f * d)*. This assumes that python’s built-in hash table is able to diminish load factors and keep the collisions from being too common.
- query lookup: There are another two loops here, one that runs once for each term in the command line query, and another that runs through each file a term occurs in, still *d*. Assigning “number of query terms” to *q* gives a rough complexity as *(d * q)*. Lookup is expected to be quick and simple, if python’s built-in process to give each key a unique location is viable.
- sorting results: Perhaps the simplest, this portion of code loops through the terms and frequencies, multiplying the result frequencies by the amount of times a term appears.

Overall, it seems that *f*, *d*, and *q* don’t factor too much into runtimes anyway. Runtimes are included in the various queries used to test the program, and queries took between 0.156 and 0.163 seconds, whether there were ten results returned or none. The system took the longest to search for “hydrotherapy” and for “international affairs,” taking 0.020 and 0.012 seconds, respectively.