#Joshua Hsin #ID 13651420
Good queries
1. A.C.M
2. machine
3. learning
4. software
5. cristina
6. lopes
7. teaching evaluation form
8. engineering
9. hello
10. uci

Bad queries
1. door hinge
2. master cristina
3. master lopes
4. uci engineering scores
5. misinformation online uci
6. peeled oranges
7. pathos plant
8. potted plant
9. universities to study abroad at
10. locomotive uci

For all queries, rather than basing scoring off the weighted word frequencies (less important words given 1 point while more important words given 1.5 points), I used the recommended tf-idf score, weighting and multiplying term frequency and document frequency.

weight = (1 + log10(tf(t,d))) * log10(N/df(t))

For queries with multiple words, I first purge stop words that have less relevance to document/url matching if they make up less than half of the query. Stop words usually have less relevance to the word score due to their high document frequency. This allows the search to be easier since the query becomes smaller and can also help the query results become more accurate.

For queries of more than two words, I first match and sort queries that match the amount of words in the query, then continue iterating and looking for urls as needed. On each iteration, I require one less word match each time (down to two minimum), sort, and add urls to the match list until I run out of words to match or reach 20 total urls. I then print the top 20 results sorted.

Finally, I improved time for queries of more than two words by keeping a tracking list of the top highest document/url values in each iteration. In the first iteration, the maximum size is 20. If the list has less than 20 elements, I append the current matching url value to the list and add the key and value of the url to the temporary dictionary. If the list has 20 elements, I check that the current document/url value is greater than the minimum of the 20 highest document/url values. If so, then I replace the value of the previous url by the new url and add the key and value of the url to the temporary dictionary. After each iteration, I sort the temporary dictionary and transfer to the final match dictionary. If I get 20 urls, I break from the loop, but if not, I keep iterating. Assuming the first iteration does not result in 20 urls, in the next iteration, I make the size of the tracking list 20 minus the current number of urls in the match dictionary. I keep going until I reach 20 urls or run out of iterations.