REPORT (Assignment 3: Milestone 3)

CS 121, May 28, 2021

TEAM: kakao: Yechan Seo, 86604330, Seungmin You, 23545234, Hyungkyu An, 48987888

Our Testing Queries

[‘Information’, ‘Alex Thorton’, ‘how to change major’, ‘Donald bren school’, ‘Counseling hours’, ‘UCI Application’, ‘CS 161’, ‘financial aid’, ‘undergraduate tuition’, ‘uci graduation’, ‘campus job’, ‘uci biology’, ‘scholarship’, ‘job opportunity’, ‘lopes cristina’, ‘steve jobs’**]**

Poor Performance and Optimization

Input Query: ‘Computer’

- We first create the tf-idf score to rank the result of queries. However, we first used the ‘sort’ method to sort the result that we have retrieved. We soon found out that our performance was way over the limited time. We decided to use the “heap” method when comparing tf-idf scores of documents within a single query.

Input Query: ‘computer science artificial intelligence’

- First, we implemented the secondary index for bookkeeping by creating a dictionary, called “indexGuide”, that holds the seek position of two prefixes of all the tokens from the inverted index (i.e., ‘ce’ = 244322). However, the overall search query performance time took over 1000 ms, and such inefficiency was caused by too much I/O access due to the structure of the secondary index. We had to readline() over a few thousand times per token because we only stored two prefixes. Though we did as such to minimize the size of the secondary index, we found, in fact, that the total number of tokens was manageable and our dictionary size was small (11KB) so that we were able to try a different approach of enlarging the secondary index to be more specified on the tokens. By memorizing all the seek positions of the index tokens, we could end up with the index size, only 770KB, and processed it 6 times faster than before (average full access time of 4-length query is 70~150ms).

Input Query: ‘information retrieval search engine’

- At first, we used the AND operation to find the documents that contained all the given query words. It seemed that we have successfully retrieved the result and returned it. However, after a few times of testing, we could realize that when we used AND operation, we were not considering that each of the query tokens can have different importance. So, we implemented

an effective system that sorts the input query based on idf score before the retrieval process (high-idf ranking method introduced in the lecture). We kept the weights of the tokens' importance and gave their corresponding priority to the combinations that consisted of more relevant and important tokens.

Input Query: 'computer szolpozojkelsasdfqw'

- As we built up the inverted index and search engine algorithm, we decided to create a search query, 'computer szolpozojkelsasdfqw,' that seemed like a typo as one of our testing query because we thought that the search engine should definitely handle unexpected typos from users. At first, we implemented our initial search engine assuming all queries are just as valid. For this reason, when the engine tried to find the query, it failed to find 'szolpozojkelsasdfqw' from the index of course because that token was a typo not in the index. So we had an error with a timeout. We then decided to exclude a token if the token is not registered in our index. In this way, we could prevent most of the unexpected typos from users when they search their query. Comparing each of the input query words with the inverted index data, we not only saved the time accessing the token that will not be retrieved but also successfully handled possible typos from users..