Project 2: MuGle

Members

1. Krittin Chatrinan ID 6088022  
2. Anon Kangpanich ID 6088053  
3. Tanawin Wichit ID 6088221

SUMMARY REPORT

# OVERVIEW

The organization of the project can be divided into 2 parts: indexing and querying. Each part contains nested inner Util Classes which provide methods that facilitates the process as shown below.

# QUESTIONS

## QUESTION A

### TF-IDF w/ Cosine Similarity VS. Jaccard Similarity

From the figure A, it can be observed that at the same k and recall value TF-IDF yields better precision and F1 values. This means that foreach k value TF-IDF search engine can retrieve more relevant documents proportional to non-relevant documents. As a consequence, TF-IDF search engine tends to serve the information need of the user better than the Jaccard one.

One of the reasons why TF-IDF with cosine similarity is better than Jaccard Similarity is that TF-IDF with cosine similarity does not take each term equally by using Inverted Document Frequency (IDF). In another words, some terms may have higher weight because of its rarity in a corpus. The other reason is that TF-IDF with cosine similarity also consider weight of each document based on the frequencies of terms that occur in the query. Lastly, by treating each document as a vector, TF-IDF with cosine similarity is also advantageous when it comes to comparing document and query. If we construct both document and query vectors, we can calculate its norm and calculate the angle between those vectors.

However, in Jaccard Similarity, it only works on binary values indicate whether a term is present or not present in a document. Moreover, Jaccard Similarity does not take term frequency and document length into account; therefore, every term and document are the same if the term presents in the query. Thus, Jaccard Similarity is inferior when it comes to relevancies returned documents.

## QUESTION B

### Precision-Recall Plot

1. **Different Precision Value at the same Recall Value** – As illustrated in figure B, it can be observed that the precision values at the same recall is significantly different between Jaccard and TF-IDF. The chart shows that TF-IDF consistently yields better precision in every recall value which means that TF-IDF’s search results has more relevant documents proportional to all retrieved documents.
2. **TF-IDF’s F1 value is consistently larger than Jaccard’s F1 value** – Due to the fact that F1 is a harmonic mean between precision and recall, in another word, it quantify both precision and recall into a value, larger F1 value means larger precision and recall. With better precision value, it means that the search engine retrieve more relevant documents proportional to all retrieved documents. Moreover, with higher recall, it means that the search engine retrieve more relevant documents proportional to all relevant documents in the corpus. Therefore, from the figure C, TF-IDF search engine, which has higher F1 overall, is a significantly better than Jaccard in terms of getting relevant documents.
3. **Increasing Recall when k is increased** – From figure D, it can be observed that Recall value on both search engines are increasing with higher k value. TF-IDF has higher growth rate as observed from the slope. Therefore, overall recall value of TF-IDF is consistently higher than that of Jaccard. However, in higher k value, both TF-IDF and Jaccard suffers from dimishing growth rate.
4. **Decreasing Precision when k is increased** – From figure D, it can be observed that the precision curve of Jaccard is initially steep which means that it has higher decrease rate than TF-IDF. Similarly to Recall, both TF-IDF and Jaccard, in high k values, has a flatter slope which can be inferred that the decreasing rate is diminishing with higher K value.

## EXTRA CREDIT QUESTION

### Comparing with MyCoolSearcher

// TODO: Introduce MyCoolSearcher

1. **Consistently Higher Precision** – Iterating through all blocks linearly consume time. Having multiple threads could speed up the process but at the expense of more ram usage and more coding. Parallelism can make this program very scabable to a very large dataset.
2. **Maintaining block-level Large data structures** – Instead of storing docDict, termDict and postingDict globally we should store them after finishing one block like binary index file to save memory when the operation runs long.
3. **Index files merging algorithm** **with Parallelism** – Using multiple threads to divide up the merging process can boost up the speed, but again with more memory used.
4. **Choosing the right file system** – Index file could get very large; therefore, choosing the right filesystem can potentially boost up the performance.

# BONUS: RANKED QUERY

### Implementation Details

We have