# Project-3 Comp 479/6791
# Fall 2020


**Dept. of Computer Science and Software Engineering**
**Concordia University**

Submitted to: - Dr. Sabine Bergler

Submitted by: Yashika Khurana
Student id: 40094722

## Introduction

## 1. Sub project 1 (SPIMI)

In this module the main focus is to build the blocks which have structure like [“block1”: [{docid1:tf},{docId2:tf),…,“block2”: [{docid1:tf},{docId2:tf),…]. So to build the indexer I have used Reuters corpus 21578 and read all the 22 .sgm file. So first the document is split up according to the <!DOCTYPE lewis SYSTEM “lewis.dtd" which is the starting line of any document. Then all the documents are extracted by the tag which is the start tag ‘<REUTERS’

‘</REUTERS>’ which is the ending tag for the document.

After that Title, date and body tag is extracted to get information about the document. Then punctuation is removed and created the tokens with the doc id e.g. (tokens, id). Then I saved the token, id pairs in the file “tokens.json”.

In the function generate_blocks(), this block is used to read the stream of file and generates the block based on the following parameters:

- For the report purposes 500 block size is taken to generate the block1.json and blocklast.json which shows the blocks generated during the process. When we used 500 as block size the no. of total blocks were 5829.
- For the implementation purpose block size is taken as 10,000 and then the total no of blocks reduced to 291.
- Then each block after generation is sorted and then save to final list which contains all the blocks (dynamic generation of all blocks till we cover the parameter value or if we reach to the end of the input token stream of token, id pair)

To calculate the time taken to generate the blocks ‘time’ module of the python is used.

In total it take average 50 seconds to build the blocks which is less then the naive indexer, which achieves the purpose of SPIMI because we divide the blocks and then do the sorting.

**Merging:**

Then all the blocks generated above will be considered for the merging to build the SPIMI unranked index. The approach used is to first find the lowest token present in all the blocks. Then that lowest token is compared with all blocks first value, so if it is present the dicid,tf will be merged. The final structure of the merging is {“term1”:[{docid1:tf},{docid2:tf},…}

The output of the final indexer is saved in the file “InvertedIndex.json” to see the final index.

## 2. Sub project 2 (Probabilistic search engine)

In this module the output of the indexer of the given query is used to rank that documents with the help of BM25 formula which is:

$$RSV_d = \sum_{t \in q} \log \left[ \frac{N}{\mathrm{df}_t} \right] \cdot \frac{(k_1 + 1)\mathrm{tf}_{td}}{k_1((1 - b) + b \times (L_d / L_{ave})) + \mathrm{tf}_{td}}$$

With the help of this documents are ranked which help us to get the relevant documents first.

To compare the result with different k and b values query “Brierley" is tested with the following parameter settings:

(“Brierley”, k=1, b=0.2) results saved in bm25_comparison_1.json

(“Brierley”, k=1, b=0.5) results saved in bm25_comparison_2.json

(“Brierley”, k=1, b=0.8) results saved in bm25_comparison_3.json

(“Brierley”, k=2, b=0.8) results saved in bm25_comparison_4.json

(“Brierley”) run on SPIMI index, results saved in spimi_comparison.json

After analyzing the different comparisons of the k and b values, the results were same except different ranking values. Therefore, further values k=1 and b =0.5 is used.

## 3. Test Queries

In this module, we are asked to design 4 different type of test queries and to show the results.

**(a) a single keyword query, to compare with Project 2:**
Queries used ["logic", "belt", "obtain", “Empire"] to compare the results with the project2. These single query term were applied on the BM25 formula to give the ranking. The results are better now with the help of this formula because it gives more meaningful doc id first. The result of single query term are stored in the file “single_term_queries.json”.

**(b) a query consisting of several keywords for BM25:**
Several keywords queries used for the analysis - [“investors”,”stock”,"Ottawa"]. The results were also verified manually to check the relevant documents first and it also matched with the BM25 formula ranked documents. The results are stored in “bm25_several_queries.json”. The output is {docid: rsv}

**(c) a multiple keyword query returning documents containing all the keywords (AND), for unranked Boolean retrieval:**
To run the AND queries, these queries are used [“earth”,”minerals”,”titanium"]. The results of these queries are also verified manually and which gives the intersection of all doc id which contains all the keyword. The result of these queries are stored in “and_queries_unranked.json”. In the output we have [docid1, docid2….]

**(d) a multiple keywords query returning documents containing at least one keyword (OR), where documents are ordered by how many keywords they contain), for unranked Boolean retrieval:**
To run the OR queries, these queries are used [“earth”,”minerals”,”titanium"]. The results of these queries are also verified manually and which gives the UNION of all doc id which contains at least one of the keyword. Then the results are ranked according to the docs which contains all the keywords in it. The result of these queries are stored in “or_queries_unranked.json”. In the output we have {dicid:”no of keywords queries it occurred in that document”,…..}

## 4. Information need

well documented sample runs for your queries on the information needs:

**(a) Democrats' welfare and healthcare reform policies:**
To process this information, first punctuation is removed from this, then this query on BM25 runs on the following logic:

First for the individual term rsv is calculated and for the common doc id, these rsv values are added before final ranking of the documents.The results of this blocks are saved for the BM25 in "information_need_query_a_bm25.json"

**(b) Drug company bankruptcies:** The queries results for the BM25 are stored in "information_need_query_b_bm25.json". First for the individual term rsv is calculated and for the common doc id, these rsv values are added before final ranking of the documents.

**(c) George Bush:** The queries results for the BM25 are stored in "information_need_query_c_bm25.json". First for the individual term rsv is calculated and for the common doc id, these rsv values are added before final ranking of the documents.

These sample when used to run with BM25, gives the good result in terms of first it gives the documents which are most relevant and then less relevant and so on, depending upon the rsv value for the particular docid.

## 5. Program structure

The program structure is divided into 1 main file, i.e. 'SubProject1.py'.

To run the project - **Please simply run the "**SubProject1.py**" python file** which calls the generate_blocks(), merge_blocks and contains the queries. All the results are saved in the file as .json format

While running the code merging take time, please give some time to do the merging.

## 6. Output files

There will be files generated by running the "main.py"

1. tokens.json:

Contains tokens, id pair

2. block1.json:

Contains block1 (term: postings list), block size 500 is used

3. blocklast.json:

Contains block1 (term: postings list), block size 500 is used

4. invertedIndex.json:

SPIMI unranked index

5. bm25_comparison_1.json, bm25_comparison_2.json, spimi_comparsion.json bm25_comparison_3.json, bm25_comparison_4.json:

Used to show comparisons between various k and b values

6. single_term_queries.json

Test queries to run single term queries

7. bm25_several_queries.json

Test queries to run several queries term on BM25

8. and_queries_unranked.json

Test queries to run and queries on unranked index

9. or_queries_unranked.json

Test queries to run or queries on unranked index

10. information_need_query_a_bm25.json

Information need query a) BM25 output

11. information_need_query_b_bm25.json

Information need query b) BM25 output

12. information_need_query_c_bm25.json

Information need query c) BM25 output