Experiment-2
Block Sort Based Indexer and Building Term-Document Matrix

***Abstract*— This lab involves dividing the text into chunks, creating posting-list corresponding to chunks, then sorting them and storing them in different text files. Then, merging all the files to get a single posting list of all the terms in the corpus.**

I. INTRODUCTION

Block sort based indexing algorithm is a solution to the problem of the big data where we can't store everything in our RAM. So, we store it in a sequential manner in chunks inside the secondary storage. The BSBI Algorithm involves:

- The collection is segmented into equal sized parts.
- the termID-docID pairs of each part is sorted in memory
- Intermediate results are stored on disk
- Finally, all intermediate results are merged into the final index.

O(Tlog(T)) is the time complexity of block sort based indexing. The highest time complexity involving step is sorting, and the upper bound for the number of items we must sort is T (i.e., the number of termID–docID pairs). But, the actual indexing time is usually dominated by the time it takes to parse the documents and to do the final merge.

II. DISCUSSION ON SOLUTION

We are using gutenberg dataset for creating posting list. First, we will import all the library which will help us for doing block sort based indexing such as stopWords, PorterStemmer etc. we will not make posting-list for stop words because stop words are the words which have no meaning so while traversing over the dataset if we found a stop words we will skip that word otherwise we will create posting-list for that word. for posting list we are storing term and document id in which that particular term appear and frequency how many time a term appear in a particular document and total frequency of that term in given dataset which is basically the sum of frequency of all the documents in which that particular term appear.for storing this posting list we are using 2D dictionary of integers, so that for each term we have a dictionary in which document id is key and value is frequency of that term in this document. we can't store the whole data in our RAM because of size issue so we have to divide our data into chunks. chunk size is number of terms and document id and if our chunk size is greater than a constant value(in our case it is 100000) we will write this posing list in a text file and clear our RAM. to write a chunk in file we sort the list on basis of term and for a particular term we have list of document id and we will sort this list in increasing order of document id. after sorting we are writing all these data into text file, we are creating different text files for different chunks and after creating all the text files we are merging all files into one file in such a way that in final file all the posting list is sorted according to term and for a particular term, list of document id in which that term appeared is also sorted in increasing order. our one final text file will look like

$$t1\ \{d1,x\}\ \{d2,y\}\ x{+}y$$

here t1 is term. d1 and d2 is document id and x and y is frequency of term t1 in document d1 and d2 correspondingly and at the end we have total frequency of that particular term in the given dataset. and this whole process is called as block sort based indexing because we are dividing our data into blocks and sorting that blocks and finally merging all the blocks.

III. INTERPRETATION

We worked out a way in which we can calculate the *tf-idf* score for all the terms in the corpus. The *tf-idf* score is a product of term-frequency and inverse document frequency.

$$idf = log(\frac{D}{d})$$

Term-frequency is the number of documents in which the term appears and the inverse document frequency is defined as

$$idf = log(\frac{D}{d})$$

where D is the total number of documents in corpus and d is the number of documents in which the term appears. So, basically if the term appears in every document of the corpus, *idf* is equal to 0. The fewer documents the term appears in, the higher the *idf* value.

We can interpret, the tf-idf score of a term gives us the relevance of a term inside the corpus.

IV. CONCLUSION

- Block sort-based indexing helps us to create posting lists for chunks of data, sort them and store them in different text files which helps to reduce the memory usage.
- Finally, we combine all the different text files together in one single file to form a **Term Document Matrix** which can be used for various purposes.
- Once we find the Term Document Matrix, we can then run different queries on the Matrix to get meaningful output.
- For ex. If we need to find the document in which terms: *Brutus* and *Dagger* both appear, we can do so with the help of Term Document Matrix.

- For small data, it can be done quickly but when we are dealing with the big data, we need to use clustering so as to minimize the output time.
- Given the term frequency of two terms to be equal, then the term which has the least document frequency, has the highest *tf-idf* score and vice-versa.

V. Links

- The code can be viewed here.
- All the Output files can be viewed here