This folder contains the source code for executing Boolean queries on inverted index and positional index.

Input Files:

Source data contains nearly 20,000 articles related to motorcycles, baseball, hockey etc arranged in 20 folders.

Total size of text data: 4.46 MB (after decompression)

Steps performed for creating inverted list:

A) *Pre-processing* :

- All the text files were broken into tokens . RegexpTokenizer was used for serving this purpose.
- The unnecessary punctuations were removed from the words.
- These tokens were converted into lower case in order to avoid two entries for the upper case of the same word.

B) *Creating inverted index* :
    - Inverted index is implemented using the python dictionary where key is referred as vocabulary and the value represents the list of documents in which this vocabulary term was present.
    - All the keys were sorted in lexicographical order
    - All the posting lists were sorted in increasing order of their doc IDs.

Processing Boolean Queries:

Code can be used to process generalized Boolean query of the form:

Query: 'A' op 'B' op 'C' …….op 'Z'

Where OP can be :

- AND
- OR
- AND NOT
- OR NOT

In any sequence.

And the output will contain three parts:

- Number of documents returned.
- List of documents returned.
- Minimum Number of comparisons performed.

Assumptions:

1) User is assumed to enter at least 2 query terms separated by any valid Boolean operator mentioned above.
2) Code might not work properly if used input some punctuations.
3) Code assumes that the query terms are valid and are not misspelled.
4) Minimum number of comparisons is subjected to the order in which Boolean operators are processed. Code has given priority to 'AND' over 'OR'. 'Not ' operator is handled in a more smarter way in order to reduce total number of comparisons. Please refer the heading "Handling Boolean Operators" for more detailed description.

Handling Boolean Operators

In this code AND , OR ,NOT are assumed to be "logical" operators.

First of all the input Boolean expression is broken down in multiple sub expression when an OR operator is encountered. For example :

'X' AND NOT 'Y' AND 'Z' OR 'K' AND 'C' OR NOT 'M'

Sub expression 1: 'X' AND NOT 'Y' AND 'Z'

Sub expression 2: 'K' AND 'C'

Sub expression 3: NOT 'M'

Each sub expression contains collection of 'AND' and 'NOT AND' operators. These operations take O(x+y) time where X and Y are sized of the posting lists of operands.

Firstly all the operands in a sub expression are sorted in the increasing order of the document frequency and then AND / NOT AND operations are performed.

The reason for giving priority to AND over OR is the size of the resulting posting list may decreases when intersection is performed.

Lastly all these resulting posting lists are OR ed in any order .

Handling phrase queries using positional index:

First of all the common inverted index is used for phrase query and Boolean queries.

All the doc IDs are retrieved which contain all these terms query terms not necessarily in a phrase form.

Once all these docs are fetched , the relative positioning of terms were checked in order to lookup only those docs in which there terms are present in the form of the phrase which was inserted by the user.

References:

https://dzone.com/articles/sorting-dictionaries-in-python

https://www.geeksforgeeks.org/

https://www.youtube.com/watch?v=aUgrfDhPc4Y&t=299s