Information Retrieval
Assignment - 1
- Ankit Mishra & Rajat Prakash

Question 1: Boolean Search

- - **Dataset loading**
  A folder Humour, Hist, Media, Food is created to store all the files from zip. Latin - 1 encoding is used on text from 1133 files and loaded into python.
- **Preprocessing**

  Using NLTK library, we have done following steps to filter and clean the textual data

  - Conversion to lowercase alphabets and tokenization
  - Replacement of contractions with actual words. We used a general dictionary filled with all the well-known contractions. Source for this dictionary : dictionary
  - Removal of stop words
  - Removal of punctuations and unwanted characters from the textual data using the library.
  - Initially, we tried stemming each token. The word belongs in the dictionary or not won't affect the search algorithms. Hence, we switched to stemming of tokens instead of lemmatization.
  - Using pickle library, we have stored all the document name, original texts, filtered and cleaned text. pickle file is generated in order to use them in future efficiently.
- **Index Creation**

  At the outset, we sorted our dictionary list by name. It helped in index creation and simplified the process. Further, a mapping of document-to-document ID is generated. It is stored for future usage. For each of the terms, we have Posting lists.

  Now, we have stored both in following two files -

  1. Index.pkl (Index Mapping pickle file)
  2. docIds.pkl (Document IDs Mapping pickle file)
- **Query Processing**
  Given 4 types of operations.
  1. OR

  2. AND

  3. OR NOT

  4. AND NOT

Each of the operations is handled by a separate function in the program.

Moreover, each posting list we have is already sorted. So, we can greedily use the merge algorithm (similar to the one in Merge Sort algorithm). It will help in reducing the number of comparisons.

The query processing will start from left and end at right. Sentences or words separated by commas are termed as Queries. Comma is used to separate the operators.

Output -

1. No. of documents matched.
2. No. of comparisons made.
3. Names of all the matched documents.

Question 2: Phrase Search

- **Dataset Loading**

  A folder Humor, Hist, Media, Food is created to store all the files from zip. Latin - 1 encoding is used on text from 1133 files and loaded into python.

- **Preprocessing**
  Using NLTK library, we have done following steps to filter and clean the textual data
  - Conversion to lowercase alphabets and tokenization
  - Word tokenization
  - Removal of punctuations and unwanted characters from the textual data using the library.
  - Removal of stop words
  - Removal of blank spaces (“ ”) and extra spaces from all tokens

  The document name, original text, cleaned and filtered textual data stored for further processing.

- **Positional Index Creation**

  Using the cleaned document, we can create the positional index. An iteration on all the cleaned documents followed by appending the name of the document with the index of the posting list. (i.e. the position at which it is stored in the posting lists)

Using pickle library, all the positional index is stored in a pickle file – positionalIndex.pkl

- **Query System**

  1. Now, at first we use the created file positionalIndex.pkl to load the positional index from the pickle file.

  2. User input taken : query

  3. Cleaning and filtering the input query using the approach mentioned for preprocessing in Q2.

  4. Splitting the query into words. Then, checking whether the word is in the postings list. We returned an empty set if it didn't exist.

  5. A pointers array will be generated. Each pointer will point to the index of the postings list for each word of our query.

  6. Further, using an iteration over all the input query words and checking the difference between each subsequent word’s index.

  7. If the difference is not equal to one or if the document is not the same for any neighbouring words, increase the pointer by value one for the word with corresponding document name or index. Else, append the document name and move to next iteration.

  8. Take the input as a single sentence from the user.

  9. Output -

  a) No. of matched documents.

  b) The names of all matched documents.