CSE508: Information Retrieval
Assignment 1

Yukti Goswami(MT21109)
Saurabh Pandey (MT21077)

Question 1

(a) (8 points) Carry out the suitable preprocessing steps on the given dataset.

Ans1.(a)

Preprocessing steps:

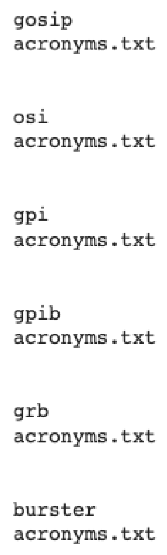
1. Reading data from files using the file path.
2. Creating sentences from these file's data
3. Splitting sentences into words
4. Applying lowercase
5. Removing punctuations and blank spaces
6. Removing stopwords
7. Applying all the above functions to preprocess the data and creating a list of unique words
8. Applying Lemmatization on these unique words

After this, we get a list of tokens of unique words from all files present in the dataset.

(b) (8 points) Implement the unigram inverted index data structure.

Ans1.(b)

For implementing the unigram inverted index data structure we have created classes such as Linked List and Node. Node class contains the frequency, docId, and pointer to the next Node, while Linked List contains a head.



```
gosip
acronyms.txt


osi
acronyms.txt


gpi
acronyms.txt


gpib
acronyms.txt


grb
acronyms.txt


burster
acronyms.txt
```

Each word is printed with its respective files in which it is present as shown in the above figure.

(c) (1+1+2+2 = 6 points) Provide support for the following queries-

Ans1.(c) For applying this we have created a merge function that consists of conditions for each operator and returns the newly formed linked list along with its total no. of comparisons.

(i) (1 point) x OR y

Ans1.(c)

(i) As the functionality of OR operator this function adds both Linked list-1 and Linked List-2 elements to the newly created Linked List.

(ii) (1 point) x AND y

Ans1.(c)

(ii) As the functionality of AND operator this function adds only when both Linked list-1 and Linked List-2 contain that element to the newly created Linked List. It also checks length and takes shorter ones for consideration.

(iii) (2 points) x AND NOT y

Ans1.(c)

(iii) As the functionality of AND NOT operator this function applies A-B operation on Linked list-1 and Linked List-2 elements and then accordingly add elements to the newly created Linked List.

(iv) (2 points) x OR NOT y

Ans1.(c)

(iv) As the functionality of OR NOT operator this function applies A or not B= U-(B-A)on Linked list-1 and Linked List-2 elements and adds elements to the newly created Linked List. Here U is the Universal set containing all words.

ASSUMPTIONS:

1. AND NOT and OR NOT are applied with a '-' sign in the middle. For example, AND-NOT and OR-NOT.
2. The query consists of words separated by spaces.

(d) (18 points) During the demo, your system would be evaluated against some queries in the format

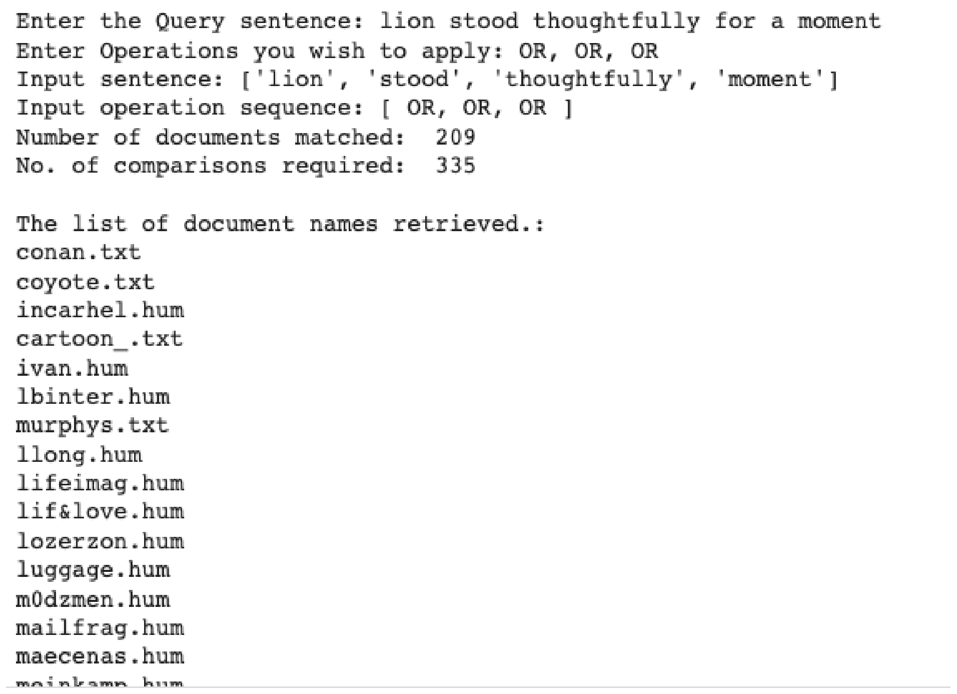
mentioned below. Marks would be awarded based on the correctness of the output.

Ans1.(d)

```
# Preprocessing the query
def pre_processing_query(query):
 file_sentences = sentenceTokens(query)
 file_strings = sentencesToStrings(file_sentences)
 file_lower = lowercase(file_strings)
 file_clean = punct_blanks(file_lower)
 file_tokens = stopWords(file_clean)
 return file_tokens
```

First preprocessing is done and then merge functionality is applied according to the input.

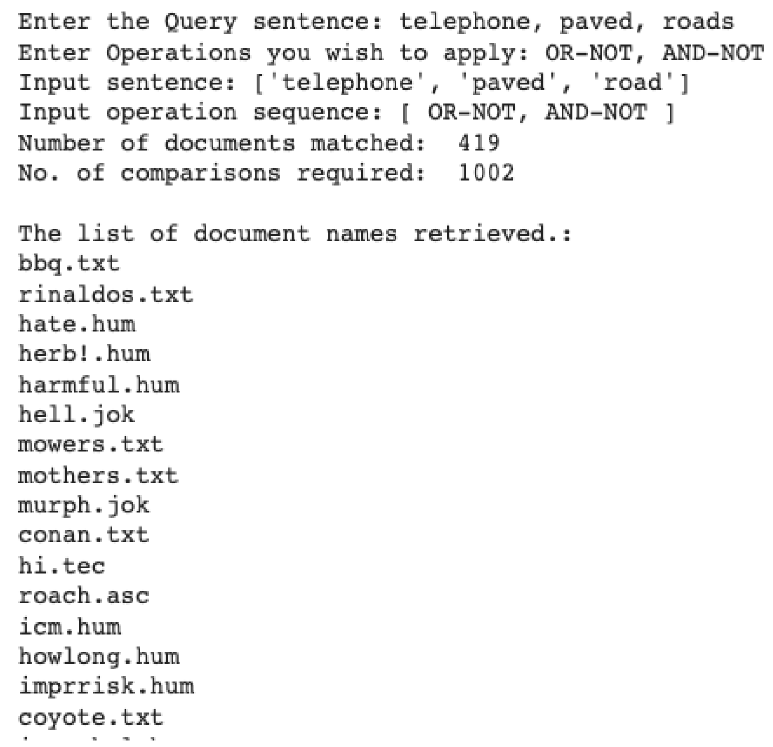
SAMPLE QUERY-1: OUTPUT



```
Enter the Query sentence: lion stood thoughtfully for a moment
Enter Operations you wish to apply: OR, OR, OR
Input sentence: ['lion', 'stood', 'thoughtfully', 'moment']
Input operation sequence: [ OR, OR, OR ]
Number of documents matched:  209
No. of comparisons required:  335

The list of document names retrieved.:
conan.txt
coyote.txt
incarhel.hum
cartoon_.txt
ivan.hum
lbinter.hum
murphys.txt
llong.hum
lifeimag.hum
lif&love.hum
lozerzon.hum
luggage.hum
m0dzmen.hum
mailfrag.hum
maecenas.hum
```

SAMPLE QUERY-2: OUTPUT



```
Enter the Query sentence: telephone, paved, roads
Enter Operations you wish to apply: OR-NOT, AND-NOT
Input sentence: ['telephone', 'paved', 'road']
Input operation sequence: [ OR-NOT, AND-NOT ]
Number of documents matched:  419
No. of comparisons required:  1002

The list of document names retrieved.:
bbq.txt
rinaldos.txt
hate.hum
herb!.hum
harmful.hum
hell.jok
mowers.txt
mothers.txt
murph.jok
conan.txt
hi.tec
roach.asc
icm.hum
howlong.hum
imprrisk.hum
coyote.txt
```

Question-2

Methodology:

- ☐ I performed the same preprocessing of data for Query which was done for the documents. This is done to ensure that the phrase query after preprocessing takes the same structure as the document data itself.
- ☐ All the documents were read and preprocessed as the requirements of the question.
- ☐ After which we got the tokens for each document.
- ☐ Each document was given a document ID and was mapped using a dictionary.
- ☐ Next, each of these tokens was picked one by one and was converted into a posting that said that the given term belongs to which document and the list all the locations in which it is present.
- ☐ This was stored as the positional index.
- ☐ Next, the query was reduced into terms after preprocessing.
- ☐ Firstly it was checked that the terms are not of length 0.
- ☐ If not, then we pick each term of the query and fetch its relevant posting from the posting index.
- ☐ In case no posting is found for a given term, it returns with a blank result since for a phrase query the primary requirement states that each word of the phrase should be present to the most possible degree.
- ☐ Similarly posting for each term was fetched.
- ☐ Then we found the intersection of all the documents that have all the terms present in the query and stored them in a list.
- ☐ Further, we picked each of these common documents and checked the position in which each word is present with respect to the next word, and the difference was expected to be of 1 only.
- ☐ All those documents were picked and were returned as the resultant list of documents that were displayed.

Preprocessing

- ☐ The preprocessing of data and the query both have been done using the same steps as given in the question.
  - Convert the text to lower case
  - Perform word tokenization
  - Remove stopwords from tokens
  - Remove punctuation marks from tokens
  - Remove blank space tokens

Assumption:

- We have assumed that we will have sufficient main memory to be able to store the entire positional index in the main memory at any given time