

1. Report:

(i) **Describe the steps you have performed for data preprocessing and provide answers** for the following questions.

(a) How many tokens does the corpus contain before stopword removal and stemming?

2859563

(b) How many tokens does the corpus contain after stopword removal and stemming?

2234287

(c) How many terms (unique tokens) are there before stopword removal, stemming, and case-folding?

54247

(d) How many terms (unique tokens) are there after stopword removal, stemming, and case-folding?

43360

(e) List the top 20 most frequent terms before stopword removal, stemming, and case-folding?

0 : the : 144424

1 : of : 73606

2 : to : 73074

3 : in : 54939

4 : and : 54545

5 : said : 53096

6 : a : 52018

7 : s : 32592

8 : for : 27307

9 : mln : 26732

10 : it : 22882

11 : 3 : 21874

12 : dlrs : 21273

13 : on : 19392

14 : reuter : 18964

15 : pct : 18046

16 : is : 16875

17 : lt : 16680

18 : that : 15527

19 : from : 15277

20 : by : 15154

21 : its : 14995

22 : will : 14855

23 : vs : 14836

24 : be : 14738

25 : at : 14517

(f) List the top 20 most frequent terms after stopword removal, stemming, and case-folding?

0 : to : 73074

1 : said : 53096

2 : s : 32592

3 : mln : 26732

4 : 3 : 21874

5 : dlrs : 21273

6 : on : 19392

7 : reuter : 18964

8 : pct : 18046

9 : It : 16680  
 10 : that : 15527  
 11 : from : 15277  
 12 : by : 15154  
 13 : vs : 14836  
 14 : at : 14517  
 15 : 000 : 13438  
 16 : year : 13109  
 17 : u : 11326  
 18 : billion : 10726  
 19 : has : 10185  
 20 : company : 9699  
 21 : cts : 9219  
 22 : would : 9200  
 23 : not : 8308  
 24 : inc : 8161  
 25 : bank : 8018

(ii) Describe the data structures that you used for representing the positional index (dictionary and postings).

It is dictionary as {tokenID : {docID : invIndexItem}}

that is keys are tokenID's and vales are dictionaries whose keys are docIDs and values are invIndexItem's. Where an invIndexItem is basically a container as (docID , list(positional indexes))

Using dictionaries give robust speed where also wastes some space but, that is not a concer for this case. (Tried dict of list too but took too, long apx 30minutes, to execute.)

(iii) Provide a screenshot of running your system for a conjunctive query.

(iv) Provide a screenshot of running your system for a phrase query.

(iv) Provide a screenshot of running your system for a proximity query.

```

me@memachine: ~/Desktop/python/Doc_Retv_Sys$ ./queryProcessor.py
q to exit
Query: 1 america AND north
[2, 79, 224, 382, 478, 652, 939, 1019, 1133, 1214, 1422, 1609, 1661, 1662, 1990, 2128, 2255, 2374, 3078, 3163, 3168, 3175, 3183, 3325, 3349, 3676, 3751, 4081, 4092, 4239, 4292, 4329, 4702, 4846, 5131, 5473, 5488, 5559, 5808, 6003, 6193, 6360, 6377, 6393, 6956, 7135, 7237, 7577, 7578, 7803, 7820, 8075, 8094, 8136, 8201, 8471, 8567, 8702, 9298, 9733, 9742, 9771, 9866, 10133, 10150, 11165, 12277, 12432, 12518, 12590, 12647, 12799, 12865, 12928, 12979, 13051, 13546, 15206, 15472, 15478, 15556, 16175, 17103, 17324, 17357, 17901, 18089, 18234, 18546, 18567, 18571, 18589, 18759, 19006, 19626, 20091, 20576, 20598, 21126, 21546, 21549, 21571]
Query: q
me@memachine: ~/Desktop/python/Doc_Retv_Sys$

me@memachine: ~/Desktop/python/Doc_Retv_Sys$ ./queryProcessor.py
q to exit
Query: 2 north america
[2, 79, 224, 382, 478, 652, 939, 1019, 1133, 1214, 1422, 1609, 1661, 1662, 2128, 2255, 2374, 3078, 3163, 3168, 3183, 3349, 3676, 3751, 4081, 4092, 4239, 4292, 4329, 4702, 4846, 5131, 5473, 5488, 5559, 6003, 6193, 6360, 6377, 6393, 7135, 7237, 7577, 7578, 7803, 8075, 8094, 8136, 8201, 8471, 8567, 8702, 9733, 9742, 9771, 9866, 10133, 10150, 11165, 12277, 12432, 12518, 12590, 12647, 12799, 12865, 12928, 12979, 13051, 13546, 15206, 15472, 15478, 15556, 16175, 17103, 17324, 17901, 18089, 18234, 18546, 18567, 18571, 18589, 18759, 19006, 19626, 20576, 20598, 21546, 21549, 21571]
Query: q
me@memachine: ~/Desktop/python/Doc_Retv_Sys$

me@memachine: ~/Desktop/python/Doc_Retv_Sys$ ./queryProcessor.py
q to exit
Query: 3 north /0 america
[2, 79, 224, 382, 478, 652, 939, 1019, 1133, 1214, 1422, 1609, 1661, 1662, 2128, 2255, 2374, 3078, 3163, 3168, 3183, 3349, 3676, 3751, 4081, 4092, 4239, 4292, 4329, 4702, 4846, 5131, 5473, 5488, 5559, 6003, 6193, 6360, 6377, 6393, 7135, 7237, 7577, 7578, 7803, 8075, 8094, 8136, 8201, 8471, 8567, 8702, 9733, 9742, 9771, 9866, 10133, 10150, 11165, 12277, 12432, 12518, 12590, 12647, 12799, 12865, 12928, 12979, 13051, 13546, 15206, 15472, 15478, 15556, 16175, 17103, 17324, 17901, 18089, 18234, 18546, 18567, 18571, 18589, 18759, 19006, 19626, 20576, 20598, 21546, 21549, 21571]
Query: 
  
```

2. Source code and executable: Commented source code and executables of your document retrieval system.

Provided as separate files.

3. Readme: Detailed readme describing how to run your program.

Program is coded and tested on Ubuntu 16.04 with python3

There is nothing unusual about it.

There are two files:

preprocess.py : takes care of forming dictionary.txt and invertedIndex.txt

queryProcessor.py : runs the queries on these two files generated by preprocess.py

Plus stemmer.py from provided website for porter stemmer.

These three files should be in the same directory.

On terminal (on the directory where the files live) :

Either change permission (chmod +x preprocess.py queryProcessor.py)

Then run them in order:

./preprocess.py (takes 33 secs or so on my machine)

./queryProcessor.py (takes 5 or 10 secs to load data)

queryProcessor will be in a infinite loop asking for queries, press q and hit enter to exit.

Write your query and hit enter to do queries. Format is the same as you wanted.

Note: <querytype> must be an integer (1 or 2 or 3) and must be the first character of the query, no white spaces etc before query type please.

PS: you can give an argument if you don't want to process all the 22 sgm files

for example: `\$ ./preprocess 3` would preprocess only 3 files `reut2-000.sgm, reut2-001.sgm, reut2-002.sgm`.

PS: SGM files should be under directory called Dataset which should be next to our .py files.