CSE 435/535 Information Retrieval (Fall 2020)
Project 2: Boolean Query and Inverted Index

Due Date: October 21st 2020, 11:59 pm (EST)

Overview

In this project, you will be given a sample input text file consisting of Doc IDs and sentences. Based on this provided input text file, your task is to build your own inverted index using the data. Your index should be stored as a Linked List in memory as the examples shown in the textbook (Refer Chapter 1 – Boolean Retrieval). Having built this index, you are required to implement a Document-at-a-time (DAAT) strategy to return Boolean query results. Your implementation should be based on Python3 only.

Input Dataset

input_corpus.txt is a tab-delimited file where each line is a document; the first field is the document ID, and the second is a sentence. The two fields are separated by a tab.

Example:

```
113257   COVID-19 and diabetes mellitus: implications for prognosis and clinical management
156757   Seroprevalence of Rodent Pathogens in Wild Rats from the Island of St. Kitts, West Indies
50439    Prevalence and genetic diversity analysis of human coronaviruses among cross-border children
```

Step 1: Build Your Own Inverted Index

Implement a pipeline which takes as input the given corpus, and returns an inverted index.

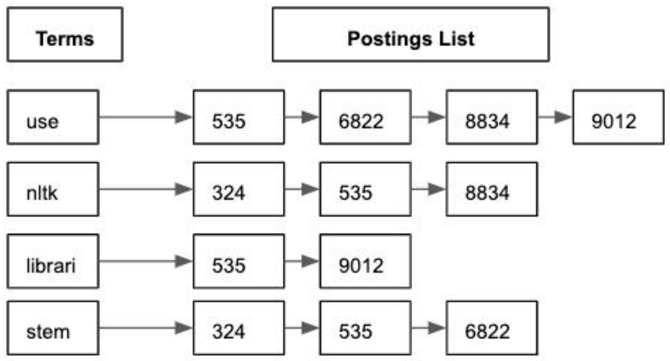
1. Extract the document ID and document from each line.
2. Perform a series of preprocessing on the document:
    a. Convert document text to lowercase
    b. Remove special characters. Only alphabets, numbers and whitespaces should be present in the document.

c. Remove excess whitespaces. There should be only 1 white space between tokens, and no whitespace at the starting or ending of the document.
d. Tokenize the document into terms using white space tokenizer.
e. Remove stop words from the document.
f. Perform Porter's stemming on the tokens.
g. Sample preprocessing output

```
Input ->  535   You can  use NLTK(library)   for stemming   !

Doc id ->  535
Doc text ->  You can  use NLTK(library)   for stemming   !

Doc text post processing ->  you can use nltk library for stemming

Doc tokens post whitespace tokenizing ->  ['you', 'can', 'use', 'nltk', 'library', 'for', 'stemming']
Doc tokens post stopword removal ->  ['use', 'nltk', 'library', 'stemming']
Doc tokens post stemming ->  ['use', 'nltk', 'librari', 'stem']
```

Note: **Do not use NLTK** or any other package for **whitespace tokenization** or **special character removal**. You might end up getting different results. Usage of simple python **regex** and inbuilt functions are recommended.

3. For each token, create a postings list. Postings list must be stored as **linked lists**. Postings of each term should be ordered by increasing document ids.



Step 2: Boolean Query Processing

You are required to implement the following methods, and provide the results for a set of Boolean "AND" queries **on your own index**. Results should be output as a JSON file in the required format.

Given a user query, the first step must be **preprocessing the query** using the same document preprocessing steps.

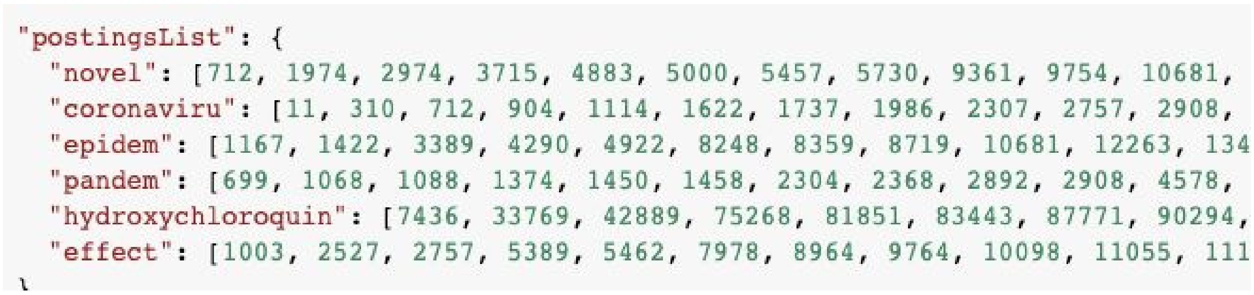
1. <u>Below are the preprocessing steps you must perform on each user query</u>

a. Convert query to lowercase
b. Remove special characters from query
c. Remove excess whitespaces. There should be only 1 white space between query tokens, and no whitespace at the starting or ending of the query.
d. Tokenize the query into terms using white space tokenizer.
e. Remove stop words from the query.
f. Perform Porter's stemming on the query tokens.
g. Sample query processing:

```
Input query ->  Which library to use for stemming?
Query post processing ->  which library to use for stemming

Doc tokens post whitespace tokenizing ->  ['which', 'library', 'to', 'use', 'for', 'stemming']
Doc tokens post stopword removal ->  ['library', 'use', 'stemming']
Doc tokens post stemming ->  ['librari', 'use', 'stem']
```

2. <u>Get postings lists</u>

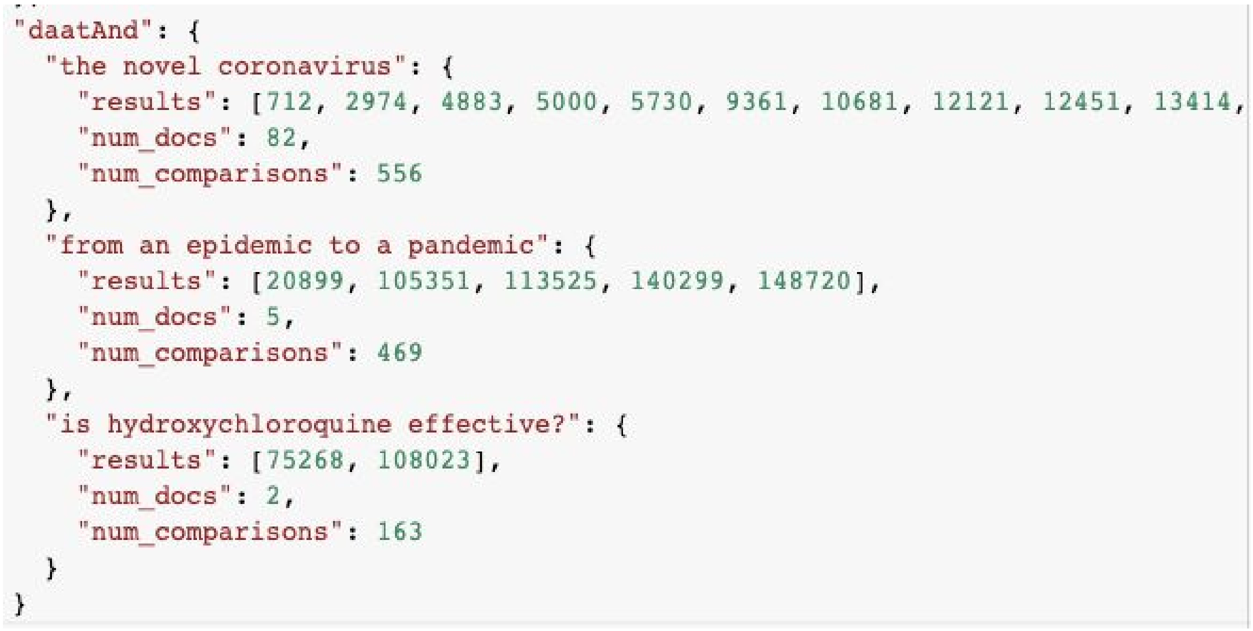
This method retrieves the postings lists for each of the given query terms. Input of this method will be a set of terms: term0, term1,..., termN. It should output the **document ID wise sorted** postings list for each term. Below is a sample format of the same:



```
"postingsList": {
  "novel": [712, 1974, 2974, 3715, 4883, 5000, 5457, 5730, 9361, 9754, 10681,
  "coronaviru": [11, 310, 712, 904, 1114, 1622, 1737, 1986, 2307, 2757, 2908,
  "epidem": [1167, 1422, 3389, 4290, 4922, 8248, 8359, 8719, 10681, 12263, 134
  "pandem": [699, 1068, 1088, 1374, 1450, 1458, 2304, 2368, 2892, 2908, 4578,
  "hydroxychloroquin": [7436, 33769, 42889, 75268, 81851, 83443, 87771, 90294,
  "effect": [1003, 2527, 2757, 5389, 5462, 7978, 8964, 9764, 10098, 11055, 111
}
```

3. Document-at-a-time AND query

This function is used to implement multi-term boolean AND query on the index using document-at-a-time(DAAT) strategy. Input of this function will be a set of query terms: term0, term1, …, termN. You will have to implement the MERGE algorithm, and **return a sorted list of document ids, along with the number of comparisons made**. Below is a sample of the output:

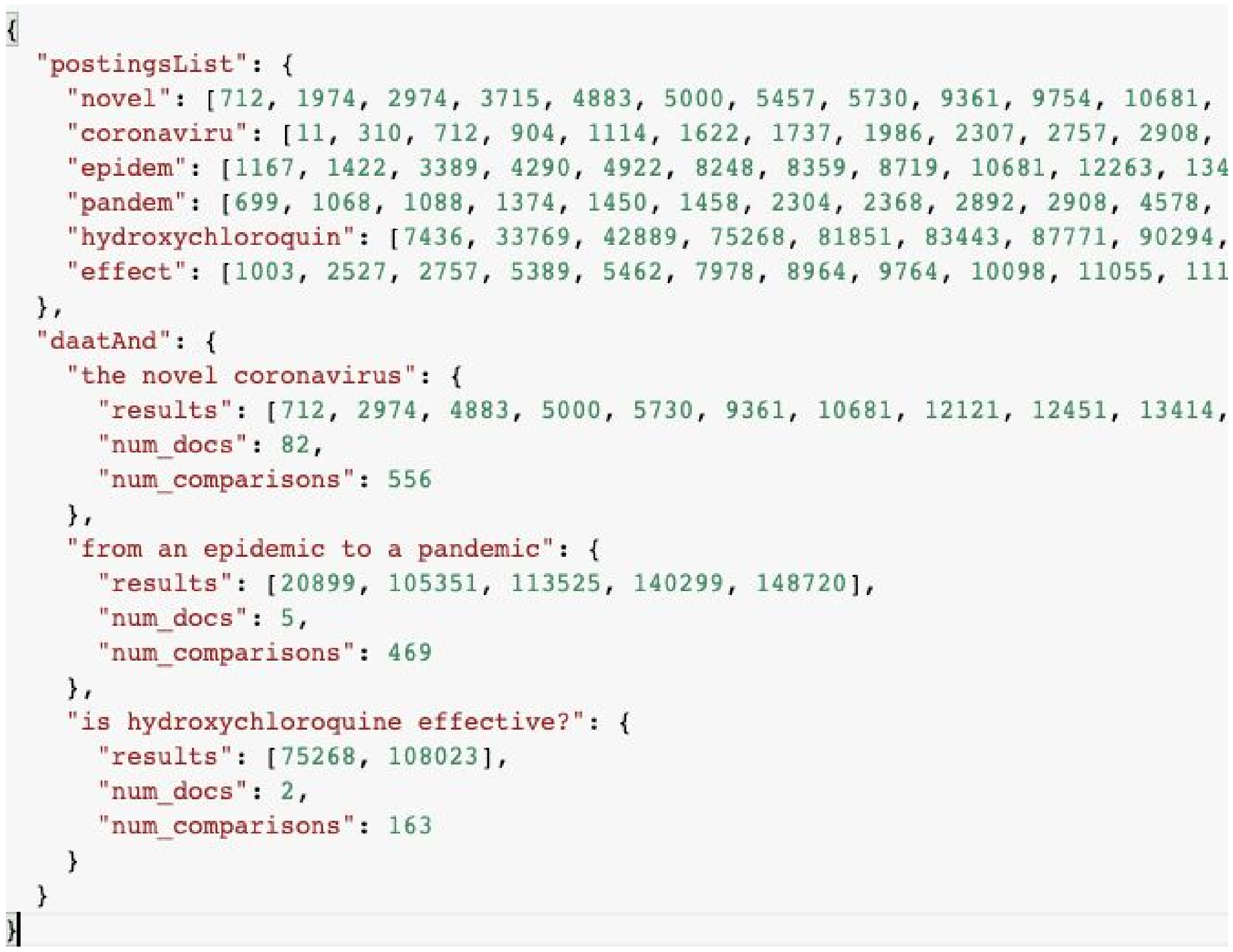


```
"daatAnd": {
  "the novel coronavirus": {
    "results": [712, 2974, 4883, 5000, 5730, 9361, 10681, 12121, 12451, 13414,
    "num_docs": 82,
    "num_comparisons": 556
  },
  "from an epidemic to a pandemic": {
    "results": [20899, 105351, 113525, 140299, 148720],
    "num_docs": 5,
    "num_comparisons": 469
  },
  "is hydroxychloroquine effective?": {
    "results": [75268, 108023],
    "num_docs": 2,
    "num_comparisons": 163
  }
}
```

<u>**NOTE**</u>: A comparison (for field "num_comparisons") is counted whenever you compare two Document IDs during the union or intersection operation. Determine the correct merge order to optimize "num_comparisons".

Step 3: JSON output in the correct format
The results of the postings list and DAAT AND queries must be combined in a single python dictionary, and saved as a json file. Below is a sample of the final output.

Sample JSON output format:



```
{
  "postingsList": {
    "novel": [712, 1974, 2974, 3715, 4883, 5000, 5457, 5730, 9361, 9754, 10681,
    "coronaviru": [11, 310, 712, 904, 1114, 1622, 1737, 1986, 2307, 2757, 2908,
    "epidem": [1167, 1422, 3389, 4290, 4922, 8248, 8359, 8719, 10681, 12263, 134
    "pandem": [699, 1068, 1088, 1374, 1450, 1458, 2304, 2368, 2892, 2908, 4578,
    "hydroxychloroquin": [7436, 33769, 42889, 75268, 81851, 83443, 87771, 90294,
    "effect": [1003, 2527, 2757, 5389, 5462, 7978, 8964, 9764, 10098, 11055, 111
  },
  "daatAnd": {
    "the novel coronavirus": {
      "results": [712, 2974, 4883, 5000, 5730, 9361, 10681, 12121, 12451, 13414,
      "num_docs": 82,
      "num_comparisons": 556
    },
    "from an epidemic to a pandemic": {
      "results": [20899, 105351, 113525, 140299, 148720],
      "num_docs": 5,
      "num_comparisons": 469
    },
    "is hydroxychloroquine effective?": {
      "results": [75268, 108023],
      "num_docs": 2,
      "num_comparisons": 163
    }
  }
}
```

<u>NOTE</u>:
There is no requirement for the names of your methods. However, you should submit a .py file **named exactly as "UBITName_project2.py"**, and your program should start running by executing the following command **on Timberlake**:

python3 UBITName_project2.py path_of_input_corpus output.json query.txt
python3 sougatas_project2.py ./data/input_corpus.txt ./output.json ./queries.txt

Here, the first input parameter *path_of_input_corpus* should be able to accept a string indicating the path of the tab-delimited file containing the corpus. The second input parameter *output.json* refers to the output file to write to, while the third, *queries.txt* refers to the input file containing the query terms.

The following assumptions can be made:

1. The number of input query terms can be varied.
2. All of the input query terms are selected from the vocabulary.
3. Query terms should be processed in the order in which they are written in the query. Say, you should process term0 first, then term1, so on and so forth.
4. DocumentID and corresponding sentence text will be separated by tab space in the input file reference by *path_of_input_corpus.*
5. **DO NOT** use python build-in methods to do unions and intersections on postings lists directly. Create your own Pointers/References!
6. We will use **Moss** to check **plagiarism**.
7. **Output should be formatted exactly the same as required. Otherwise you will not be able to get credits because grading will be automated!**

Grading and Evaluation

A successful implementation should be able to support all the functions mentioned before. It should also generate correct results in the required format. Failure in following the instructions and requirements will result in 0.

Rubrics:

Total points for this project: **10**

1. Correct output formatting: **1**
2. postingsList total score: **3**
    a. Successfully rebuild index and correct output for GetPostings: 3
3. DaatAnd total score: **6**
    a. Correct documents retrieved for DaatAnd: 3
    b. num_comparisons within acceptable range of +/- 5% of desired comparisons: 3

What to Submit

You should use cse-submit script to submit a .py file named exactly as "**UBITName_project2.py**". Double check and make sure your submission can be executed successfully **on Timberlake**. You need to use either "submit_cse435" (undergrad) or "submit_cse535" (grad), depending on your registration status.

<u>**NOTE**</u>: Late submissions will NOT be accepted. The deadline is firm (i.e., October 21st 23:59 PM (EST)), if your timestamp is 12:00 AM, it is a late submission. Please start early.

FAQ's:

1. **How should I get started for this project?**

   - First, make yourself familiar with fundamental concepts such as dictionary, postings, Inverted Index and Boolean operations. The best place to start is reading thoroughly the lecture slides and Chapter 1,2 of the referred textbook.

2. **What programming concepts are needed to complete this project?**

   - You are expected to know how to work with functions and be familiar with basic data structures such as dictionary/hash maps, lists/arrays, Linked lists, Queue and so on. Also check out the python NLTK package. You should use the functions provided in NLTK for stemming and stopword removal.

3. **My program takes a while to execute. Would that be a problem in grading?**

   - Although the focus of this project is to test the correctness, we encourage you to be mindful of the data structure you are using for implementation. Unless it takes an unreasonably long time, you are fine in terms of grading but again please carefully analyze your code.