ASSIGNMENT 3

In this assignment you will create inverted index on provided corpus.

# Corpus:

The corpus contains 3495 documents of type HTML. The documents are divided in three sub-directories (1-3).

You cannot change the names and directories of these documents.

# TASK 1: Preprocessing:

The first step in creating an index is tokenization. You must convert a document into Tokens suitable for indexing. Your tokenizer should follow these steps:

1. Accept a directory name as a input, and process all files found in that directory
2. Extract the document text with an HTML parsing library, ignoring the headers at the beginning of the file and all HTML tags
3. Split the text into tokens (You can use libraries available in python to do so)
4. Convert all tokens to lowercase (this is not always ideal, but indexing intelligently in a case-sensitive manner is tricky)
5. Apply stop-wording to the document by ignoring any tokens found in this list (stop words given in stoplist.txt on cactus)
6. Apply stemming to the document using any standard algorithm – Porter, Snowball or KStem stemmers are appropriate. You should use a stemming library for this step.

The resulting tokens will be used in next step

# TASK 2: Creating Inverted Index:

## Step 1:

Receive the token in <term, docID> pair from step2 and create the inverted index in memory using dict in python.

The corpus is divided in three subdirectories. You should treat each sub-directory as a block, you will create an index on one block at a time (you will be penalized if you create index of all document in one go).

## Step 2 Writing index to file:

Your index will be written in two files, one file for terms and one file for posting list.

Write <term>, <posting file offset> in index\_terms.txt file

Write the posting lists of each term in index\_postings.txt file

Where <posting file offset> is the byte position in index\_postings.txt file where the posting list of <term> starts.

### Posting list:

Posting list will contain the df of terms and then documents ID and position of term. It should be sorted on base of docID and further on positon, and converted into gap list using delta encoding. To get bonus you can further encode gap list using V-Bit or Vbyte encoding.

Example of posting list is given in section 5.4.5 in Bruce Croft.

For example if term tropical occurred in document 1 at position 1, 7 and 6 and in document 2 at position 6,17, and197, at in document 3 at position 1 then its posting list will be

3,1, 2, 1, 6, 1, 3, 6, 11, 180, 1, 1, 1

Where 3 is the document frequency

Green numbers are document ID/gap between document ID

Blue numbers are term frequency of tropical in each document

Black numbers are positions gaps of term occurrence in documents.

Step 1 and 2 will be repeated for each directory therefore, at the end of this step you should have 6 files named as index\_1\_terms.txt, index\_2\_terms.txt, index\_3\_terms.txt, index\_1\_postings.txt, index\_2\_postings.txt, index\_3\_postings.txt where 1, 2, 3 represent the name of subdirectory on which index was created.

# TASK 3: Merging Inverted Indexes:

In this part you will write a module that will take file path of indexes and will merger them in one index. The merged index will be written on file in same format as individual index.

You will use buffer readers and writers to perform this task, complete index file should not be read/write at once.

Use this module to merge the three indexes create in previous step. Final index should be on a file named

inverted\_index\_terms.txt and inverted\_index\_postings.txt

# TASK 4: Keeping additional Information:

As required in previous step, each document should have a unique integer docID be. docID should be assigned to the document at the time of preprocessing. You will also need to store document length which will be used in next assignment (Ranked retrieval).

Keep a separate file that maps sub\_directory/documentName to docID and its document length, named docInfo.txt. Format of this file should be as follow.

docID,sub\_directory/documentName,documentLength,magnitudeofDocument.

Not that magnitude and length are two different things. Magnitude is used in cosine similarity formula (|V(D)|) and length is BM25 (length d)

# TASK 5: Reading Inverted Indexes:

In this task you will perform Boolean retrieval. You will get a query as input, tokenize the query in same way as in task 1.

Your program should print to the list of documents containing the query terms, one document file name on each line in ascending lexicographical order in following format

Sub\_directory/documentName

If no result is found print

No result found

This module should also work on both indexes (compressed and uncompressed).

# Submissions:

Submissions will be on google classroom, you will only submit your code (one.py file no jupyter notebook).

# Implementation details:

You are free you use any library to implement your work.

All the mandatory requirements of assignment are given above, you can make valid assumption for any detail that is not given above.