This is the README file for A0140713U-A0196982H-A0139744W-A0140585E's submission

# == Python Version ==

I'm (We're) using Python Version <2.7.15> for

this assignment.

# == General Notes about this assignment ==

## Pre-processing:

## Indexing:

1. The pre-processed file is pickled as a dictionary of documents (Case ID) that are each a dictionary (“title”, “content”, “date\_posted”, “court”) containing 4 lists of words in the order they are found in the document.
2. Each dictionary of document is then parsed through the method getGrams which creates 1, 2 and 3 grams from these lists and store them into a dictionary.
   1. Get grams also notes the TF of each gram for the individual document
   2. After performing getGrams for the whole document, the output is a dictionary of title, content, date\_posted and court which are a dictionary containing all the grams in that part of the document and the TF in this part
3. The Length of the document is then calculated using LNC length.
   1. Each document is at this point issued an arbitrary numerical index and the Case ID and document length are stored in the main\_dictionary[“DOC\_ID”][index] = tuple((Case ID, Length)).
4. The local dictionary and local postinglists are then updated to reflect the addition of this document
   1. If the word is not in the local dictionary, add the word with DF of 1.
   2. Add a new posting list entry
5. After a certain point, the postinglists are written onto the disk and the local dictionary is added to the main dictionary. This is to unload the memory consumed. However, it also takes a very long time to do so.
   1. For this project, this is only done once at the end.
6. Posting lists are stored on the disk as a string of numbers ending with a “\n”. This is to allow the whole posting list to be read using readline().
   1. Each posting is a string of numbers containing information in the following order. Length of Index(2 digits), Index(Length digits), TF in title (1 digit), Length of TF in content (1 digit), TF in content (Length digits), TF in date (1 digit), TF in court (1 digit)
   2. Example docIndex: 50 contains 1 frequency of this word in the title. Resulting posting 🡪 025011000
7. The dictionary is stored on disk in JSON format for quick loading when running search.

## Free-Text Search:

The free-text search retrieves documents deemed relevant to the query based on the Vector Space Model and outputs the all relevant documents in decreasing order of relevance.

A posting contains the document ID and the term frequencies in the court name, title, date and content. These tf are handled separately. The scoring scheme implemented for the content and date calculates the score of relevance based on the lnc.ltc score and further awards 1 more point for each successful word match in the query. However, for docID, court and title, a simple score of 2 is given per match in the query. The scores for each of these are stored in a list in the order [docID, court, title, date, content] as deemed by order of descending important. The lists are then sorted by their natural order to reflect the relevance of the document to the query. Thus, the reason why docID, court and title are not subjected to tf-idf is because a discrete categorisation is required for the tiered ranking system to work optimally. Only the date and the content are continuous since they need to differentiate between 2 documents.

## Boolean Search:

Distinction between Boolean search and free-text search is made over the presence of the Boolean operator “AND”. They are handled respectively, and order is maintained to ensure query and output lines match one another.

Once a query line has been determined to be Boolean search, it is split into its constituent queries as delimited by “AND”. Every constituent query is treated as a mono/bi/trigram appropriately. Each query the undergoes lnc.ltc as covered in lecture and previous assignments. Each document is stored in a list, ordered by lexicographically increasing docID, together with its respective lnc.ltc score. AND-merging is then performed on all constituent query terms, with the resultant score being the product of its two scores. A product score was chosen over a sum score to retain the effect of cosine normalisation even through combining scores. The final list is then ranked by decreasing score before being written to output.

# == Files included with this submission ==

index.py

source code of indexing phase

search.py

source code of searching phase

dictionary.txt

output files from the indexing phase

postings.txt

output files from the indexing phase

README.docx

# == Statement of individual work ==

Please initial one of the following statements.

[X] I, A0140713U-A0196982H- A0139744W-A0140585E, certify that we have followed the CS 3245 Information

Retrieval class guidelines for homework assignments. In particular, we

expressly vow that we have followed the Facebook rule in discussing

with others in doing the assignment and did not take notes (digital or

printed) from the discussions.

[ ] I, A0000000X, did not follow the class rules regarding homework

assignment, because of the following reason:

<Please fill in>

I suggest that I should be graded as follows:

<Please fill in>

# == References ==

<https://docs.python.org/2/library/sets.html>

<https://docs.python.org/2/library/json.html>