INFORMATION RETRIEVAL

Assignment 1: Dayta, Dahta?

Instructions:

- Enter your group information in the linked excel sheet.
- Use a good Unicode text editor (such as BabelPad) to view Urdu files. Here is a link to the Unicode Urdu code page.
- You can download an Urdu keyboard from this website (http://www.cle.org.pk/index.htm). I'd suggest exploring this site a little so you can get a feel of the work being done on Urdu.
- Make a report in PDF format, clearly mentioning question/ part number against your answers.
- Plagiarism of any sort is not acceptable and will be severely punished.
- Late submission will not be entertained.
- **Submit your report, docids.txt (zipped), termids.txt (zipped), term_index.txt and code as one compressed file (.zip, .rar) on Google Classroom. The name of file should be your roll numbers i.e. <Roll#1_Roll#2>.zip**
- Deadline to submit this assignment is: **Sunday, 1st March, 2020, 11:59 pm.**

Update 1: You have a choice between Urdu and English. If you choose English, you can use the dataset provided on google classroom. However, if you choose Urdu and collect your own dataset by crawling you'll get bonus marks (No bonus marks if you use someone else's dataset).

Update 2: The tasks are mostly the same as in the previous version, however I've made a few changes that are required to process the provided dataset, so make sure that you read the updated description of the tasks.

Task 1: Tokenizing Documents

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

- Accept a directory name as a command line argument, and process all files found in that directory.
- Extract the document text and split the text into tokens (You can use some library for regular expression matching. To learn about regular expressions go to this link http://www.rexegg.com/regexquickstart.html).
- Convert all tokens to lowercase (this is not always ideal, but indexing intelligently in a case-sensitive manner is tricky)
- You may need to remove punctuation and numbers.
- Prepare a list of stop words from the dataset, and apply stopping. (This will require some intelligent decision making on your part as to how many stop words you should remove).
- Apply stemming to the document using any standard algorithm – Porter, Snowball, and KStem stemmers are appropriate. You should use a stemming library for this step.
- Your tokenizer will write two files:
    - docids.txt – A file mapping a document's filename (without path) to a unique integer, its DOCID. Each line should be formatted with a DOCID and filename separated by a tab, as follows:
      1234\t32435
    - termids.txt – A file mapping a token found during tokenization to a unique integer, its TERMID. Each line should be formatted with a TERMID and token separated by a tab, as follows:
      567\tapple

Task 2: Inverted Index

After tokenization, construct an inverted index.

- Create a file term_index.txt – An *inverted index* containing the file position for each occurrence of each term in the collection. Each line should contain the complete inverted list for a single term. Each line should contain a list of DOCID,POSITION values. Each line of this file should contain a TERMID followed by a space-separated list of properties as follows:

  347 1542 567 432,43 456,33 456,41

  o 347: TERMID
  o 1542: Total number of occurrences of the term in the entire corpus
  o 567: Total number of documents in which the term appears

o 432: Document Id in which term appears
o 43: Position of term in document 432

- In order to support more efficient compression you must apply *delta encoding* to the inverted list. The first DOCID for a term and the first POSITION for a document will be stored normally. Subsequent values should be stored as the offset from the prior value. Instead of encoding an inverted list like this:

  347 1542 567 432,43 456,33 456,41

  You should encode it like this:

  347 1542 567 432,43 24,33 0,8

  Note that in order to do this, your DOCIDs and POSITIONs must be sorted in ascending order.

NOTE: You are required to construct the index by sorting termid-docid pairs as discussed in class.

Task 3: Reading the index

Now that you have an inverted index of the corpus, you'll want to be able to do something with it. This is mostly left for the next assignment. For now, we will just write the code to pull up some statistics from the index. Write a program which implements the following functionality: Passing *--term TERM* will stem the term and then list the following term information. For example:

./read_index.py --term apple
Listing for term: apple
TERMID: 342
Number of documents containing term: 58
Term frequency in corpus: 75

We will evaluate your program by running these commands for selected documents and terms.

Task 4: Report

- Explain the pre-processing steps that you've applied and how you've implemented them, which libraries you've used.
- How many stop words you've removed and how it impacted the size of the index.

- Generate word - frequency graph for the corpus. Words would go on the horizontal axis and frequencies on the vertical. (You can plot this graph in excel by exporting words and their frequencies to an excel sheet or by writing a program).
- Generate word - log_frequency graph for the corpus. Words would go on the horizontal axis and log_frequencies (use any base e.g. $\log_{10}$) on the vertical (You can plot this graph in excel by exporting words and their frequencies to an excel sheet or by writing a program). Why are there big stairs towards the end of the graph? What are the differences between the two graphs? Report your answers and graphs.