(http://www.flickr.com/people/jacalynsna a/)

(http://www.comp.nus.edu.sg)

Information Retrieval

NUS SoC, **2015/2016**, Semester II Video Conferencing Room (COM1 02 VCRm) / Fridays 11:00-13:00

Last updated: Tuesday, February 17, 2015 01:09:39 AM SGT - Added two hints also posted to IVLE forum

Homework #2 » Boolean Retrieval

In Homework 2, you will be implementing indexing and searching techniques for Boolean retrieval described in Lectures 2 and 3.

Indexing

Your indexing script, `index.py` , should be called in this format:

```
$ python index.py -i directory-of-documents -d dictionary-file -p postings-file
```

Documents to be indexed are stored in directory-of-documents. In this homework, we are going to use the Reuters training data set provided by NLTK. Depending on where you specified the directory for NLTK data when you first installed the NLTK data (recall that the installation is triggered by `nltk.download()`), this data set is located under a path like:

```
.../nltk_data/corpora/reuters/training/
```

On sunfire, recall that we installed the NLTK python module and its corpora under the class UNIX account. So the Reuters training data is here:

```
/home/course/cs3245/nltk_data/corpora/reuters/training/
```

Recall that the dictionary is commonly kept in memory, with pointers to each postings list, which is stored on disk. This is because the size of dictionary is relatively small and consistent, while the postings can get very large when we index millions of documents. At the end of the indexing phase, you are to write the dictionary into `dictionary-file` and the postings into `postings-file` . For example, the following command writes the dictionary and postings into `dictionary.txt` and `postings.txt` .

```
$ python index.py -i /home/course/cs3245/nltk_data/corpora/reuters/training/ -d dic
tionary.txt -p postings.txt
```

Although you can use any file names as you like, in this homework please follow the above command to use `dictionary.txt` and `postings.txt` , so that our marking script can easily locate the files.

In order to collect the vocabulary, you need to apply tokenization and stemming on the document text. You should use the NLTK tokenizers (`nltk.sent_tokenize()` and `nltk.word_tokenize()`) to tokenize sentences and words, and the NLTK Porter stemmer (`class nlkt.stem.porter`) to do stemming. You need to do case-folding to reduce all words to lower case.

Skip Pointers

You also need to implement skip pointers in the postings lists. Implement the method described in the lecture, where *math.sqrt(len(posting))* skip pointers are evenly placed on the a postings list. Although implementing skip pointers takes up extra disk space, it provides a shortcut to efficiently merge two postings lists, thus boosting the searching speed.

Searching

Here is the command to run your searching script, `search.py` :

```
$ python search.py -d dictionary-file -p postings-file -q file-of-queries -o output
-file-of-results
```

`dictionary-file` and `postings-file` are the output files from the indexing phase. Queries to be tested are stored in `file-of-queries` , in which one query occupies one line. Your answer to a query should contain a list of document IDs that match the query in increasing order. In the Reuters corpus, the document IDs should follow the filenames (that is, your indexer should assign its document ID 1 to the filename named "1"; also note that while Reuters doc IDs are unique integers, they are not necessary sequential). For example, if three documents 12, 40 and 55 are found in the search, you should write "12 40 55" into `output-file-of-results` in one line. When no document is found, you should write an empty line. The results in `output-file-of-results` should correspond to the queries in `file-of-queries` .

Your program should **not** read the whole postings-file into memory, because in practice, this file may be too large to fit into memory when we index millions of documents. Instead, you should use the pointers in the dictionary to load the postings lists from the postings-file. Make sure you use the `seek` and `read` I/O functions that come from python's IO library for this.

The operators in the search queries include: `AND` , `OR` , `NOT` , `(` , and `)` . The operators will always be in UPPER CASE (lower case "and"s, "or"s and "not"s simply won't occur in your data (but you should probably bulletproof your code anyways). You can safely assume that there is no nested parentheses, for example, the query `(a AND (b OR c))` will not occur. However, there only a light restriction on the length of the query (won't be over 1024 characters but can be long). Note that parentheses have higher precedence than `NOT` , which has a higher precedence than `AND` , which has a higher precedence than `OR` . `AND` and `OR` are binary operators, while `NOT` is a unary operator. Below is an illustration of a valid example query:

```
bill OR Gates AND (vista OR XP) AND NOT mac
```

While indexing is an off-line phase, searching is designed to be real-time (the extreme example is Google Instant), thus efficiency is very important in searching. In this homework, we won't be evaluating based on how fast your program can **index** a list of documents (the preprocessing), but we will test the efficiency of your searching program (runtime speed), as well as its accuracy.

What to turn in?

You are required to submit `index.py` , `search.py` , `dictionary.txt` , and `postings.txt` . Please do not include the Reuters data.

Essay questions

You are also asked to answer the following essay questions. These are to test your understanding of the lecture materials. Note that these are open-ended questions and do not have gold standard answers. A paragraph or two are usually sufficient for each question. You may receive a small amount of extra credit if you can support your answers with experimental results.

1. You will observe that a large portion of the terms in the dictionary are numbers. However, we normally do not use numbers as query terms to search. Do you think it is a good idea to remove these number entries from the dictionary and the postings lists? Can you propose methods to normalize these numbers? How many percentage of reduction in disk storage do you observe after removing/normalizing these numbers?
2. What do you think will happen if we remove stop words from the dictionary and postings file? How does it affect the searching phase?
3. The NLTK tokenizer may not correctly tokenize all terms. What do you observe from the resulting terms produced by `sent_tokenize()` and `word_tokenize()` ? Can you propose rules to further refine these results?

Submission Formatting

You are allowed to do this assignment individually or as a team of two. There will be no difference in grading criteria if you do the assignment as a team or individually. **updated** For the submission information below, simply replace any mention of a matric number with the two matric numbers concatenated with a separating dash (e.g., A000000X-A000001Y).

For us to grade this assignment in a timely manner, we need you to adhere strictly to the following submission guidelines. They will help me grade the assignment in an appropriate manner. You will be penalized if you do not follow these instructions. Your matric number in all of the following statements should not have any spaces and any letters should be in CAPITALS. You are to turn in the following files:

- A plain text documentation file `README.txt` : this is a text only file that describes any information you want me to know about your submission. You should not include any identifiable information about your assignment (your name, phone number, etc.) except your matric number and email (we need the email to contact you about your grade, please use your u*******@nus.edu.sg address, not your email alias). This is to help you get an objective grade in your assignment, as we won't associate matric numbers with student names.
- All source code. We will be reading your code, so please do us a favor and format it nicely.
- A plain text file `ESSAY.txt` that contains your answers to the essay questions.

These files will need to be suitably zipped in a single file called `<matric number>.zip` . Please use a zip archive and not tar.gz, bzip, rar or cab files. Make sure when the archive unzips that all of the necessary files are found in a directory called <matric number>. Upload the resulting zip file to the IVLE workbin by the due date: 4 Mar 2016, 11:59:59 pm SGT. There absolutely will be no extensions to the deadline of this assignment. Read the late policy if you're not sure about grade penalties for lateness (grading.html#late).

Grading Guidelines

The grading criteria for the assignment is tentatively:

- 40% Correctness of your code
- 10% Documentation
- 30% Evaluation: we will test the accuracy and querying speed of your searching program
- 20% Essay questions

Disclaimer: percentage weights may vary without prior notice.

Hints

- Indexing a large number of documents may take a while. We suggest that you just test and develop your system on a subset that will make it quick to do experimentation. For example just use the first 10 or 100 documents in your development and debugging. Also, since indexing all 7,000+ documents may take a while, please ensure you save enough time to actually do the queries.
- In the same as spirit as above, make sure you have a working solution to one part before modifying it to do another part. Save your incremental, working progress in another (directory/file/source control system) before starting the next part. In particular, if you work with just a few documents first, you may want to do the tasks in this order:
    1. In-memory indexing of a few documents, testing the Boolean operators for correctness.
    2. Implementing the stemming, tokenization methods described.
    3. Implementing postings disk-based indexing. Here you can separate the search and indexing parts into two files as suggested.
    4. Implementing the skip pointers.
- You'll need to use the python (lower-)level file input/output commands, `seek()` , `rewind()` , `tell()` and `read()` . You must use these operations when coding `search.py` , and not rely on other modules (e.g., `linecache`).
- For indexing, as we are not concerned with run-time efficiency here, you may use other modules.

One Python module to look at is `linecache`. Please look through the documentation or web pages for these.

- How do you check the correctness of your results? Use your peers! Suggest a few queries that you ran your system on, and post them and the results that you get to the IVLE forum (make sure you use the right topic heading).
- Be aware of any differences in data type sizes on sunfire vs. your development machine. To ensure the portability of your code, check and record the size of the data types in a variable and seek/read accordingly. We will not be responsible if your code works fine on your development machine, but not on sunfire.
- The Reuters corpus has some particularities (as does practically all corpora). There are some character escapes that occur in the corpus (`&lt; for <`) but you can safely ignore then; you do not have to process these in any special way for this assignment.
- Shunting-yard: For parsing those nefarious Boolean expressions, you may want to turn to the father of algorithms, Edsger Dijkstra (Min not so humbly thinks that all Computing students should know how to spell his surname). See http://en.wikipedia.org/wiki/Shunting-yard_algorithm (http://en.wikipedia.org/wiki/Shunting-yard_algorithm).
- The Pickle and cPickle library may come in handy for loading and saving data structures to disk. You might check out http://pymotw.com/2/pickle/ (http://pymotw.com/2/pickle/).
- Similarly, bulletproof your input and output. Make sure directories (e.g., arguments to `-i`) are correctly interpreted (add trailing slash if needed). Check that your output is in the correct format (docIDs separated by single spaces, no quotations, no tabs).
- Finally, you will not be provided with any skeleton code for this assignment, and for subsequent ones. You should consider copying (or downloading again) the template for Homework 1 and modifying it appropriately for this assignment.