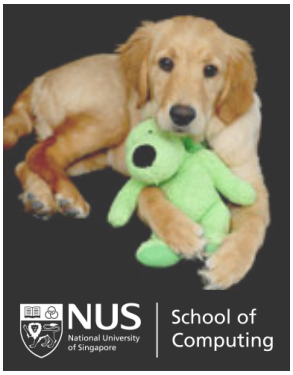

Information Retrieval

NUS SoC, AY 23/24, Semester II, Fridays 12:00-14:00

Last updated: 9 Feb 2024 - Information updated for AY 23/24 Semester 2

Deadline for submission: **4 Mar (Mon), 2pm SGT**

Homework #2 » Boolean Retrieval

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

(Note: the sample commands below are meant to illustrate how we will run your system for evaluation. Do **not** 1) hardcode the paths and filenames in your program, or 2) introduce your own command options / arguments that need to be used to run your system.)

Indexing

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

```
$ python3 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/
```

(Note: If you have countered some errors running `nltk.download()`, please try `nltk.download('reuters')` instead and post in the forum if the errors persist.)

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 the 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`.

```
$ python3 index.py -i nltk_data/corpora/reuters/training/ -d dictionary.txt -p p
```

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 nltk.stem.porter`) to do stemming. You need to do case-folding to reduce all words to lower case.

Scalable Index Construction

To make your index construction process scalable, you need to implement BSBI or SPIMI as introduced in the lecture. Do set a suitable block size or a memory limit in your system so that it creates two or more blocks and merges them into one eventually.

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 postings list. The storage and use of skip pointers must be explicit (e.g., you can't just store math.sqrt(len(posting) and use array arithematic for the jump).

Although implementing skip pointers takes up extra disk space, it provides a shortcut to efficiently merge two postings lists, thus boosting the searching speed.

Since you are also required to implement BSBI or SPIMI, you should be reading in the posting lists one by one after the index has been constructed, add the skip pointers accordingly and then write out the updated index.

Searching

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

```
$ python3 search.py -d dictionary-file -p postings-file -q file-of-queries -o ou
```

`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 are no nested parentheses, for example, the query `(a AND (b OR c))` will not occur. However, there is 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 `README.txt`, `index.py`, `search.py`, `dictionary.txt`, and `postings.txt`. Please **do not include the Reuters data or the sanity queries.**

Restrictions:

In general, you are allowed and encouraged to use NLTK for text processing. However, since the objective of this homework is for you to implement a Boolean retrieval system for practice, you are **not** allowed to use NLTK or any similar packages / libraries for the implementation of 1) core indexing logic, 2) core query processing logic, and 3) skip pointers.

Marks will be deducted for violations of these restrictions.

If you are unsure whether certain packages / libraries can be used, please use the forum to clarify.

Essay questions:

You are also encouraged to think about the following essay questions as you work on the assignment. These questions are meant to enhance your understanding of the lecture materials and the assignment. You are not required to submit your answers but you are welcome to discuss your answers with us. Note that these are open-ended questions and do not have gold standard answers.

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.

If you are forming a team for this assignment, for the submission information below, simply replace any mention of a student number with the two student numbers concatenated with a separating dash (e.g., A000000X-A000001Y). Only one student in the team needs to make the submission on the team's behalf. For multiple submissions received from the same team, we will mark only the most recent submission.

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 student 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` (e.g., in UPPERCASE): this is a text only file that describes any information you want me to know about your submission. You should give an overview of your system and describe the important algorithms/steps in your system. However, You **should not** include any identifiable information about your assignment (your name, phone number, etc.) except your student number and email (we need the email to contact you about your grade, please use your *e*******@u.nus.edu* address, not your email alias). This is to help you get an objective grade in your assignment, as we won't associate student numbers with student names.
- All source code. Minimally, you should submit your `index.py` and `search.py` . However, please feel free to include additional files as needed. **We will be reading your code, so**

please do us a favor and format it nicely.

- All files related to your index. Minimally, you should submit your `dictionary.txt` and `postings.txt`. However, please feel free to include additional files as needed.
- (Optional) A plain text file `ESSAY.txt` that contains your answers to the essay questions (if you choose to attempt the questions). Again, do not disclose any identifiable information in your essay answers.

These files will need to be suitably zipped in a single file called `<student 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 `<student number>`.

A package of skeleton files have been released in Canvas to help you prepare the submission. You should use 1) the **sanity check queries** in the package to ensure that your system is robust and does not crash even when unseen / weird inputs are given, and 2) the **homework checker script** to check whether your zip archive fulfills the criteria above. Upload the resulting zip file to Canvas by the due date: **4 Mar (Mon), 2pm 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 Guidelines

The grading criteria for the assignment is tentatively:

- 50% Correctness of your code
- 20% Documentation
  - 5% For following the submission instructions and formatting your documentation accordingly.
  - 5% For code level documentation.
  - 10% For your high level documentation, in your README document.
- 30% Evaluation: we will test the accuracy and querying speed of your searching program
- 0% Essay questions
- Note: Nominal bonus marks (+1) might be given to submissions that excel in the above-mentioned components.

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? Exchange queries with your peers! Suggest a few queries that you ran your system on, and post them and the results that you get to the Piazza forum (make sure you use the right topic heading).
- Be aware of any differences in data type sizes on the Compute Cluster 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 the SoC Compute Cluster.
- The Reuters corpus has some particularities (as does practically all corpora). There are some character escapes that occur in the corpus (`<` 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. See http://en.wikipedia.org/wiki/Shunting-yard_algorithm.
- The Pickle library may come in handy for loading and saving data structures to disk. You might check out https://docs.python.org/3.7/library/pickle.html.
- 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).

Back to top