Information Retrieval (CS 317)

Fall 2019

Programming Assignment 1

In this assignment, you will write an indexer and use it to index a collection of documents. In the next assignment, you will create a ranker that uses your index to process queries. If you combine these two assignments, you will have constructed a simple search engine. This is individual assignment. For this assignment, you will be using document collections at this path

\\ Cactus \xeon\Maryam Bashir\Information Retrieval\corpus

Step 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:

1. Accept a directory name as a command line argument, 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 some library for regular expression matching. To learn about regular expressions go to this link http://www.rexegg.com/regex-quickstart.html)
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 (\\Cactus \xeon\Maryam Bashir\Information Retrieval\stoplist)
6. 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.
7. 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`

Step 2: Inverted Index

- 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`
  - 347: TERMID
  - 1542: Total number of occurrences of the term in the entire corpus
  - 567: Total number of documents in which the term appears
  - 432: Document Id in which term appears
  - 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 should create inverted index using two different methods.

1. **Using Hashmaps or dictionary**
2. **Without Hashmaps or dictionary (by sorting termid-docid pairs)**

Step 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 command line interface. Your program must not scan the inverted index linearly; it must look up the offset in term_info.txt and jump straight to the correct inverted list.

Keep in mind as you design this program that you will be reusing much of this code in the next assignment.

You can call the program anything you like, and in Java your command will look slightly different. Note that the values in the output examples below are made up.

Passing `--term TERM` will stem the term and then list the following term information:

```
$ ./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.

Submission on GitHub

Create a private repository on Github for this assignment and add your section's TA as collaborator. Section A TA is Nashit and Section B TA is Ibrahim Zafar. Their Github usernames will be shared soon.

You should make incremental commits on Github for this assignment. Number of commits will contribute in marks of this assignment.

Submission Checklist:

- Part 1: Your source code
- Part 1: docids.txt (zipped or gzipped)
- Part 1: termids.txt (zipped or gzipped)
- Part 2: Your source code (Using HashMap)
- Part 2: Your source code (Without HashMap)
- Part 2: term_index.txt (zipped or gzipped)
- Part 3: Your source code

Rubric [Total 20 marks]

Inverted Index source code (Using HashMap) [5 Marks]

Inverted Index source code (Without HashMap) [5 Marks]

Correct Format of term_index.txt [5 Marks]

Delta Encoding [3 Marks]

Read_index code is working [2 Marks]