Programming Assignment 1

Overview

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.

You will write three programs:

1. A tokenizer, which reads a document collection and creates documents containing indexable tokens
2. An indexer, which reads a collection of tokenized documents and constructs an inverted index
3. A tool which reads your index and prints information read from it

Part 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 a parsing library, ignoring the headers at the beginning of the file and all HTML tags if any
3. Split the text into tokens
4. Apply stop-wording to the document by ignoring any tokens found in this list (Stopword list will be provided)
5. Apply stemming to the document using any standard algorithm. You can use a stemming toolkit for this (UNLT https://eprints.lancs.ac.uk/id/eprint/165032/).
6. Your tokenizer will write three 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\doc123`
    - 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\tasparagus`
    - doc_index.txt – A *forward index* containing the position of each term in each file. One line of this file contains all the positions of a single token in a single document. Each line should contain a DOCID, a TERMID, and a list of all the positions of that term in that document (the first term has position 1, the second has position 2, etc.). The DOCID, TERMID, and positions should be separated by

tabs, as follows:
`1234\t567\t1\t3\t12\t42`

Part 2: Inverting the index

The final steps in index construction are inverting the index and preparing for fast random access to terms' inverted lists. Write a program which reads doc_index.txt to produce the following files.

- 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. That is, it should contain a TERMID followed by a list of DOCID:POSITION values. However, 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:
  `567\t1234:9\t1234:13\t1240:3\t1240:7`
  you should encode it like this:
  `567\t1234:9\t0:4\t6:3\t0:4`
  Note that in order to do this, your DOCIDs and POSITIONs must be sorted in ascending order.
- term_info.txt – A file that provides fast access time to the inverted list for any term in your index, and also provides extra metadata for the term. Each line of this file should contain a TERMID followed by a tab-separated list of properties:
  `567\t1542\t567\t315`
  - 1542: The offset in bytes to the beginning of the line containing the inverted list for that term in term_index.txt. If you jump to this location and read one line, the first symbol you see should be the TERMID.
  - 567: The total number of occurrences of the term in the entire corpus
  - 315: The total number of documents in which the term appears

Part 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 just `--doc DOCNAME` will list the following document information:

```
$ ./read_index.py --doc clueweb12-0000tw-13-04988

Listing for document: clueweb12-0000tw-13-04988
DOCID: 1234
Distinct terms: 25
Total terms: 501
```

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

```
$ ./read_index.py --term asparagus

Listing for term: asparagus
TERMID: 567
Number of documents containing term: 315
Term frequency in corpus: 567
Inverted list offset: 1542
```

Passing both `--term TERM` and `--doc DOCNAME` will show the inverted list for the document/term:

```
$ ./read_index.py --term asparagus --doc clueweb12-0000tw-13-04988

Inverted list for term: asparagus
In document: clueweb12-0000tw-13-04988
TERMID: 567
DOCID: 1234
Term frequency in document: 4
Positions: 134, 155, 201, 233
```

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

Submission Checklist:

Submit the complete code on Google classroom as zip file.

- Part 1: Your source code
- Part 1: docids.txt (zipped or gzipped)
- Part 1: termids.txt (zipped or gzipped)
- Part 1: doc_index.txt (zipped or gzipped)
- Part 2: Your source code
- Part 2: term_index.txt (zipped or gzipped)
- Part 2: term_info.txt (zipped or gzipped)
- Part 3: Your source code

Rubric

- **Part 1: 2 Points**
  - Your tokenization algorithm is correct: you produce the correct stream of tokens from a given HTML document
  - The contents of docids.txt are correct

    - The contents of termids.txt are correct
    - The contents of doc_index.txt are correct
- **Part 2: 2 Points**
    - You correctly construct the inverted index
    - Your inverted lists are correctly delta-encoded
    - The contents of term_index.txt are correct
    - The contents of term_info.txt are correct
- **Part 3: 1 Point**
    - Looking up a document works correctly
    - Looking up a term works correctly
    - Looking up an inverted list works correctly

In case of plagiarism, negative marks will be given.

Good Luck!