**Information Retrieval**

In this task, we have written an indexer and use it to index a collection of documents. In the next task, we will create a ranker that uses your index to process queries. If we combine these two tasks, we will have constructed a simple search engine.

**Step 1: Tokenizing Documents**

The first step in creating an index is *tokenization*. We’ll convert a document into a stream of tokens suitable for indexing. Following steps will be included:

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.
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 (List attached in repository.)
6. Apply stemming to the document using any standard algorithm – Porter, Snowball, and KStem stemmers are appropriate.
7. The tokenizer will write two files:
8. 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

1. 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

1. 347: TERMID
2. 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, we’ll 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 will 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

* It will be encoded like this:

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

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

**Note: We’ll 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 we have an inverted index of the corpus, we’d be able to do something with it. This is mostly left for the next task. For now, we will just write the code to pull up some statistics from the index. We’ll write a program which implements the following command line interface. The program won’t scan the inverted index linearly; it must look up the offset in term\_info.txt and jump straight to the correct inverted list.

We can call the program anything, and in Java the 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

**Files attached hereby:**

* The source code
* docids.txt
* termids.txt
* term\_index.txt
* stoplist
* corpus