National University of Computer and Emerging Sciences, Lahore Campus

	Course:	**Information Retrieval**	**Course Code:**	**CS 4051**
	Program:	**BS(Computer Science)**	**Semester:**	**Fall 2021**
	Due Date	**23-Oct-2021 at 11:59 pm**		
	Section:	**CS**		
	Type:	**Assignment 1**		

Important Instructions:

1. Plagiarism will result in negative marks.
2. Late submission of your solution is not allowed.
3. You can make groups of two students for this assignment.
4. The solution should be submitted by only one of the group members.

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. For this assignment, you will be using a corpus of web documents provided to you.

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 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 the list of stop words provided to you.
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 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\tclueweb12-0000tw-13-04988\n`

- termids.txt – A file mapping a token found during tokenization to a unique integer, it's TERMID. Each line should be formatted with a TERMID and token separated by a tab, as follows: `567\tasparagus \n`
- doc_index.txt – A *forward index* containing the frequency (count) of each term in each file. Each line should contain a DOCID, a TERMID, and the frequency of TERMID in the given DOCID. The DOCID, TERMID, and frequency should be separated by tabs, as follows: `1234\t567\t30\n`
  The doc index should be primarily sorted by doc id and secondarily sorted by term id.

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.

1. term_index.txt – An *inverted index* containing the postings list of each term. Assume that a term having id=690 appears in two documents having ids 3 and 100. Also assume that the term appears 9 times in doc (id=3) and 2 times in doc (id=100). Then, the postings list for this term will be:
   `690\t3:9\t100:2\n`
   The term ids will be sorted by ascending order in the inverted index. The postings list of a term will be sorted by document id.
2. term_info.txt – A file that provides fast access 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\n`
   - 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 information about the term with respect to the given document.

```
$ ./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
```

Submit the following in a zipped file:

- Part 1: Your source code
- Part 1: docids.txt
- Part 1: termids.txt
- Part 1: doc_index.txt
- Part 2: Your source code
- Part 2: term_index.txt
- Part 2: term_info.txt
- Part 3: Your source code