Assignment 2: Collection Index Construction

**Instruction**

The goal of the assignment is to develop practical understanding of constructing searchable index for a document collection, which is specified as a set of classes. The students can take advantage of the outcomes of their assignment 1 (the pre-processed corpus) as the starting point of this assignment. However, you may use the pre-processed corpus we provided (see below). You need to follow the instructions to implement the required classes.

The collection index is based on the inverted file structure discussed in the class. Usually, it has at least the following two components:

* Dictionary term file. This file should contain all the index terms, their collection frequency (i.e., how many times this term appear in the whole collection), and a pointer to their corresponding posting information in the posting file.
* Posting file. This file should contain the corresponding pointer that can link entries in dictionary term file to that in the posting file. This file also includes a repeated set of information that indicates the document id that the term is in, the term frequency (i.e., how many time this term appears in this document), and other information you may want to put into the postings.

Besides, in indexing procedure, you need to assign the integer docid to each document, and your index structure should be able to transform string docno into integer docids, which is necessary for later search process (your Assignment 3).

**Tasks**

### Task 1: Build an index.

In this task, you should implement:

* **Indexing. PreProcessedCorpusReader**

You will need to get access to the result.trectext and result.trecweb, and return document one by one through the nextDocument(). Reference HW1Main class in assignment 1 for the format of the document. If you do not want to use your own result.trectext and result.trecweb, you can DOWNLOAD the versions we provide to you from Canvas.

* **Indexing.MyIndexWriter**

This class has one essential method **IndexADocument (String docno, String content)** to create index for a document represented by the **docno** and the **content**. The content is a list of words, segmented by blank space generated in the Assignment 1. You will need to write very efficient code in this class, otherwise your memory may fail to support your code. If your computer have a memory smaller than 8G, we strongly suggest you to construct the index by installments, where each installment works on only a block of the documents to be indexed. For example, each block can have n document (n can be 10000, 20000, etc). When processing the documents in a block, everything about the index can be stored in the memory, then when all the documents in the block is processed, the corresponding dictionary and postings can be stored as separate files on the hard drive so that the memory is cleaned for the next block of documents. Once all the blocks have been processed, there will be a fusion process to merge all the dictionary files, and all the posting files. If you are not clear on this process, consult the part of the slides. In this way, small memory computer can process big corpus.

You need to develop your own data structures used for dictionary term file and posting files. But **we strongly suggest you first look at task 2 before implementing task 1**, so that you can check whether your index can support search tasks in task 2.

### Task 2: Retrieve posting lists of tokens from an index

In this task, you should implement:

* **Indexing. MyIndexReader**, which has the following methods:
  + **MyIndexReader()**: read the index file you generated in task 1. **Do not load the whole index into the memory.** A proper implementation should load the dictionary term file first. Once the corresponding links of the posting information of the query terms are known, you can load the relevant parts of the postings into the memory.
  + **int GetDocid( String docno )** and **String getDocno( int docid )**: provides transformation between string docnos and integer docids.
  + **int[][] GetPostingList( String token )**: retrieve posting list of the token as a 2-dimension array (see comments in MyIndexReader for the structure of the array)
  + **int GetDocFreq( String token )**: get the document frequency of the token.
  + **long GetCollectionFreq( String token )**: get the collection frequency of the token.

**HW2Main** class is the main class for running your tasks. You can find the class in src.zip, and you are **NOT** allowed to change anything in this file. If you have successfully implemented the classes in task 1&2, you should be able to directly run HW2Main.

**Classes.Path** contains addresses of all input and output files, so you should put all files in the corresponding directory. Path.java/Path.py has been updated with two more addresses, so please replace the old one with this new one. You are **NOT** allowed to change anything in this file, too.

The classes to be implemented can be found in src.zip. You **CANNOT** change the classes’ names or the required methods’ names. However, you can add new variables, constants, and methods in these classes and create new classes if necessary.

**Requirements and Reminders**

* You CAN ONLY use Java or Python to finish this assignment.

For Java:

* You CANNOT use any Java API library other than the standard JDK (for example, you cannot use apache commons, apache Lucene, indri etc. in this assignment).
* Feel free to use IDE tools such as Eclipse and Netbeans, Spyder, etc.
* Do tell us the Java version you used for writing your assignment, e.g. JDK 1.6 or 1.7. Please only use Oracle JDK or openjdk.

For Python:

* Use python 3 if you plan to use python to finish the assignment.
* You CANNOT use any Python library other than the built-in libraries.
* Do tell us the python version you used for writing your assignment, e.g. python 3.x

**Grading**

Your submission will be graded based on:

* Correctness of the implementation on the required functions (70%). Hint: start with result.trecweb, since it is much smaller than result.trectext.
* Efficiency of your implementation, make sure your code finish processing two collections within 5 minutes (20%).
* Necessary comments for your codes and instructions for your invert index design (10%).

## Submission Requirements

A zipped file packaged with the naming convention as “pittids\_2140a2”. For example, suppose the CSSD id is jud1, then the submission package should be jud1\_2140a2.zip.

The file package should contain:

1. src folder, which contain all your codes.
2. A short instruction on how to read your scripts and how to run your scripts, including environment configuration.
3. A short instruction on how long it takes to finish running your code, and the retrieval result of token “acow” in “trecweb”, and “yhoo” in “trectext”. You will **NOT** upload your result file, so make sure your code finish running within 5 minutes; if cannot, you must make it clear whether the code will generate the result, and how much time it costs to finish running.
4. Please write your instruction in **txt files**.