Programming Assignment #2 (due in 2 weeks – day 7 of Module 3)

Inverted Files

Inverted files are the primary data structure to support the efficient determination of which documents contain specified terms. The objective of this assignment is to process the *Headlines* corpus from the course web page, much as in the first assignment, but this time you must also build an inverted file that contains a postings list for each dictionary term. Your implementation should model a useable real-world indexing system; in particular, this means that: (a) your inverted file structure must be written to disk as a binary file; (b) your dictionary must be written to disk[1]; (c) for each word in the lexicon you must store a file offset to the corresponding on-disk posting list; and finally, (d) you should process the source text file only once (many real-word collections are so big that the cost of multiple scans is prohibitive).

Construction (60 points)
Process the collection and create the dictionary and inverted file. The docids are the ID fields in the <P> tags of the corpus. You may use more than one program for this assignment if you prefer, but only process the original source text one time. For example, you could write a program that reads the input file and writes out records like "apple doc143 2" and "orange doc4 3" to represent the fact that *apple* occurs in document 143 twice and *orange* occurs in document 4 three times. These records must be sorted (by term, then by docid) and you might use a separate program just for this. Finally, you have to write out the sorted entries as an inverted file. It may not be easiest to use three programs as in this example; however, in real-world systems, there is usually at least a scan that produces a temporary file and a merging of sorted temporary files. The reason for this is because typically records for the entire collection will not fit in the memory of a single machine. **However**, since our electronic text is small it is possible to create an index using a single program without relying on auxiliary storage for this assignment. This can be achieved by following the memory-based inversion algorithm in the notes (*Algorithm A*) and directly writing out the postings after all documents (*i.e.,* paragraphs) have been read.

For full credit your inverted file should be a binary file[2]. I suggest as a baseline, 4-byte integers for document ids and 4-byte integers for the document term frequency. This is easy, and will let your code work on larger files. However, 4-bytes is more representation than you need for these files. I suggest that you store the length of the postings list (*i.e.,* also known as document frequency) with the other information for an indexing term in your dictionary data structure.

Note, the assignment is to build a **non**-positional index as described in Chapter 4 of IIR and in the lecture notes; you should ignore term position within the documents.

Description (10 points)
Provide up front, a brief written description of what you did and the format of your lexicon and your inverted file. You should also provide:

(1) Program output that indicates the number of documents, size of the vocabulary (*i.e.,* number of unique terms), and total number of terms observed in the collection (this same information was also required for the first program).
(2) The file sizes for your dictionary and the inverted file (in bytes). Is your index (dictionary + inverted file) smaller than the original text? Which takes up more space, the dictionary or the inverted file?
(3) Output for the test cases requested in the "Testing" section (below).

---

[1] Note, the *dictionary* does not have to be written as a binary file, though it would not be unusual to do so.

2 If you choose not to make your inverted file binary, there is a 15 point deduction. Use primitive functions that write integers; you should not use built-in serialization packages (e.g., pickle in Python; writeObject in Java) for the postings lists. For example, if using Java, consider the class RandomAccessFile and the writeInt method in DataOutputStream for the inverted file. In Python, you might find the struct and io libraries helpful.

Testing (30 points)
Demonstrate the ability to identify which documents a word occurs in and the number of times that the word occurs in each. For full credit do this by reading from your binary inverted file after you've created the index. Doing this in a completely separate program would make it clear that you are in fact loading information from disk.

1. Print out the document frequency and postings list for terms: "Heidelberg", "cesium", "Trondheim", "crustacean".

2. Give document frequency, but do **not** print postings for the words: "Hopkins", "Stanford", "Brown", and "college" (these postings lists are longer -- I just want to see the counts).

3. Print out the docids for documents that have both "Elon" and "Musk" in the text. You can do this by finding the postings list for each term, and then intersecting the two lists. For this test you do not need to use or supply frequency information. Please print the docids in increasing order.

Looking ahead

- For the third programming assignment you will be given queries with the goal of ranking documents using a similarity metric such as the vector cosine method. To succeed on that assignment, it is absolutely crucial that you are able to reload a lexicon from disk and retrieve a postings list from the inverted file for any specified term.
- Assume that the dataset on the third assignment will be larger (e.g., up to 2 GB of text). I have a Java solution for Program #2 that runs in about 10 seconds. If your program takes a long time (say more than 15 minutes) you are probably doing something that is computationally inefficient.

Submission
Reminder: your submission for Program #2 should be a single PDF file that contains the description and testing output requested above, and your source code. Please make sure your name is clearly visible on the first page.