CECS 429 - Homework 4

Develop a disk-based boolean search engine.

I will give you three files:

1. `IndexWriter`: I have written this class for you. Given an existing `PositionalInvertedIndex`, this class writes three files to disk in order to support a disk-based indexing system:
   (a) `vocab.bin`: contains all the vocabulary terms from the corpus, concatenated into a single ASCII-encoded file.
   (b) `postings.bin`: contains the postings data for the terms, using the format shown in lecture. For each term, 4 bytes are written for the *document frequency* of the term, and then a sequence of 4-byte integers follow, one each for the document IDs in the term's postings list. The IDs are encoded as *gaps*, not as raw IDs. **Positions are not written to disk... yet.**
   (c) `vocabTable.bin`: a table for mapping vocabulary terms to postings byte offsets, as shown in lecture. This file consists of a sequence of pairs of 8-byte integers; the first value of each pair points to a byte offset in `vocab.bin`, indicating the location where the string value of the term exists; the second value points to a byte offset in `postings.bin` containing the postings data for the term.
2. `DiskInvertedIndex`: an adaptation of `PositionalInvertedIndex` to use the set of files created by `IndexWriter` to return postings lists, rather than an in-memory data structure. Has methods `getPostings(String term)` (now returns `int[]` instead of a List structure) and `getDictionary()`, like the naive index.
3. `DiskEngine`: has the foundations for a command-line user interface to the disk index. Allows the user to select between *building* an index for a folder and *querying* an index for a folder. Uses `IndexWriter` to build an index and `DiskInvertedIndex` to query an index.

You need to do these things in this assignment:

1. Look through the code and make sure you understand what is going on. Start with `IndexWriter`, then `DiskInvertedIndex`, then `DiskEngine`. You will need to adapt this system for your second Project milestone.
2. Augment `IndexWriter` so that it also writes term frequencies and positions to disk for each posting. Follow the format from class, using 4 bytes for each value, and writing positions as gaps.
3. The method `getPostings` in `DiskInvertedIndex` is incomplete. You must finish the method according to the instructions in the comments... otherwise the program won't work :).

Print and turn in ONLY the completed DiskInvertedIndex class.

If you are ambitious, you can try the following additions. These features will need to be incorporated into your next project milestone, so you might as well start now :).

1. Incorporate positions into the `IndexWriter` (for each document ID that you write, follow with the number of positions for that document, and then each of the positions in turn) and `DiskInvertedIndex` (when reading document IDs, you will need to read the number of positions for each document ID, and then `seek` past 4 bytes $\times$ the number of positions).
2. Change `DiskInvertedIndex.readPostingsFromFile` to return an array of `DiskPosting` objects, where a `DiskPosting` is a document ID, a term frequency, and an array of integer positions which may be null/empty.
3. Add a parameter `boolean withPositions` to `readPostingsFromFile`. If that parameter is false, then perform your existing logic: read the term frequency for a posting but don't actually read the positions. (Why would you sometimes not want to read positions?) If it is true, then read the positions into an array and include that array in your `DiskPosting` object.
4. Add a method `getPostingsWithPositions` to your `DiskInvertedIndex` class, that returns a list of `DiskPosting` objects with their positions, for use with phrase queries.