CECS 429 - Homework 3

Develop a code module that creates an on-disk representation of an inverted index's postings.

Here are my recommendations:

1. Borrow your indexing code from Milestone 1, which produces a `PositionalInvertedIndex` variable.
2. Create a class `DiskIndexWriter` with a single method `writeIndex`. You should pass your index variable, as well as the absolute path to save the postings file. (I recommend creating a new folder named **index** inside the corpus path.) `writeIndex` should:
   (a) Open a new file called "postings.bin" in binary write mode.
      i. The `DataOutputStream` class in Java has convenient methods like `writeInt` and `writeLong` that will write a variable in a binary format. The C# equivalent is `BinaryWriter` and its overloaded `Write` methods. [1] For Python, `open` a file in `"wb"` mode and use the `write` method along with `struct.pack` to write the bytes for a value.[2]
   (b) Retrieve the sorted vocabulary list from the index.
   (c) For each term in the vocabulary:
      i. Write df_t to the file as a 4-byte integer.
      ii. Retrieve the index postings for the term.
      iii. For each posting:
         A. Write the posting's document ID as a 4-byte **gap**. (The first document in a list is written as-is. All the rest are gaps from the previous value.)
         B. Write $\text{tf}_{t,d}$ as a 4-byte integer.
         C. Write the list of positions, each a 4-byte gap. (The first position is written as-is. All the rest are gaps from the previous value.)
      iv. Repeat for each term in the vocabulary.
3. `writeIndex` should return a list of (8-byte) integer values, one value for each of the terms in the index vocabulary. Each integer value should equal the byte position of where the postings for the corresponding term from the vocabulary *begin* in `postings.bin`. For example, if `writeIndex` returns the list `{0, 12, 100, 140}`, then the postings for the first term (alphabetically ordered) in the vocabulary starts at byte position 0 in postings.bin; the second term starts at byte 12, the third at byte 100, etc.
   (a) You can typically always consult an output stream variable to determine the current **position** of the output within the file. This will help you determine byte positions of where terms start in each file – and if that fails, you can try and count this by hand.

[1] Note that the C# class will write variable in **little-endian order**; ask me what this means if you don't know.

[2] Python will write values in little-endian if the system you're running on uses little endian (most/all? modern operating systems). Ask me if you want to change this.