

Homework #2 (due in 2 weeks)

Inverted Files (70=50+20 points)

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 *CAESAR-POLO-ESAU* corpus, much as in the first assignment, but this time you must 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 your inverted file structure must be written to disk, your dictionary must be written to disk, for each word in the lexicon you must store a file offset to the corresponding posting list, and finally, you should process the raw text collection only once (many real-world collections are so big that the cost of multiple scans is prohibitive).

Construction (50 points)

Process the collection and create the dictionary and inverted file. You may use more than one program for this assignment if you prefer, but only process the raw text once. For example, you may have a program that reads the input file and writes out records like “apple doc1432 2” and “orange doc4 3” to represent the fact that *apple* occurs in document 1432 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. Since our electronic text is small, it is possible to create an index using a single program without relying on auxiliary storage. 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.

Your inverted file should be a binary file¹. I suggest as a baseline, 4-byte integers for document ids and 4-byte integers for the document term frequency (for this collection, 2-byte integers may suffice). I suggest you store the length of the postings list (*i.e.*, document frequency) with the other information in your dictionary – it will be useful in HW #3.

In addition to submitting your source code, you should: (1) Give an brief overview of what you did; (2) Explain the format of your lexicon and your inverted file; (3) Report the number of documents, number of unique terms, and total number of terms in the collection (This information was also required on the first assignment); and, (4) Report the file sizes for your dictionary and the inverted file (in bytes). Is your index smaller than the original text?

For HW #3 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 crucial that you be able to reload a lexicon from disk and retrieve a postings list for any specified term.

Testing (20 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 back in your binary file formats.) Print out the document frequency and postings list for terms: “Francisco”, “midway”, and “paddy”. Give document frequency, but do not print postings for the words “Kremlin”, “KGB”, and, “Khrushchev”.

Questions (10 points apiece)

[1] This question may require a bit of research; as far as I know the topic is not treated in the text. Set difference can be computed in linear time using hashing. Describe how set difference can be computed using a brute force algorithm in $O(m \times n)$ time and more efficiently in $O(m + n)$ time; m and n are the cardinalities of the two sets. Show an example computing $\text{SetDiff}(\text{set1}, \text{set2})$ using $\text{set1} = \{\text{red, orange, yellow, green, blue, indigo, violet}\}$ and $\text{set2} = \{\text{red, blue, yellow}\}$. The correct result is $\{\text{violet, green, indigo, orange}\}$. Recall that sets are unordered. Note: no coding is required for this problem.

¹ If using Java, consider the class `java.io.RandomAccessFile` and the `writeInt` method in `java.io.DataOutputStream`

[2] Express the numbers {8, 12, and 513} three ways: using a 16-bit binary representation, and using the gamma and delta codes discussed in class.

[3] Suppose when evaluating a 10-word query with the vector space model (cosine, with tf/idf term weighting) the number of documents to be scored is enormous (like 1 billion). Describe a scheme to find 100 very good documents without fully traversing every posting entry in every term's posting list?

For More Fun (extra credit, up to 5 points)

[2 points] Using any method you prefer, find as many different one-letter off variants (i.e., misspellings or OCR errors) of the word 'Soviet' as you can that appear in the *CAESAR-POLO-ESAU* corpus. 'Sovict' would be one example.

[3 points] Use string edit distance (see IIR 3.3.3) with unit-cost insertion, deletion, and substitution to find and report all of the words from your lexicon that have edit distance ≤ 2 for 'Khrushchev'