NYU Tandon School of Engineering
Computer Science and Engineering
CS 6913, Fall 2023

October 5, 2023

Assignment #2 (due October 19)

In this assignment, you are asked to write a program that creates an inverted index structure from a set of downloaded web pages. Since the crawlers from the previous assignment are fairly slow, you will be provided a larger set of pages that you can use for this assignment; see the course web page for more information.

In the following, we will discuss the requirements for your program. Note that the data may be larger than the memory size, and thus you MUST use I/O-efficient algorithms for index construction. Also, the data is provided in a certain format that you have to figure out how to deal with. Here are some suggestions and requirements:

(0) **Languages:** It is recommended that you use C/C^{++} or Java for this project, but you may also *ask for permission* to use another language. You may also use a mix of languages. Make sure you know how to "operate on a binary level" when it comes to input, output, or compression in your language. Do not build an index by simply shoving all the posting data into one of the high level data structures provided by many languages! Also, do not load the posting data into a relational database.

(1) **Disk-based Index Structures:** You need to create an inverted index structure, plus structures for the lexicon and for the page (or docID-to-URL) table. At the end of the program, all structures need to be stored on disk in some suitable format. You may use either binary or ASCII data formats for these structures, though for full credit your disk-based inverted list structures should be in binary format at the end. It is also expected to add at least some basic index compression method to decrease the index size. (To simplify debugging, your program could have a compile flag that allows you to use ASCII format during debugging and binary format for better performance.

(2) **Problem Decomposition:** It is strongly recommended to implement this homework as 2 or 3 components, say one for generating intermediate postings from the documents and writing these postings out in unsorted or partially sorted form, one for sorting or merging the postings, and one for reformatting the sorted postings into the final index and lookup structures (though this last part can be combined with the sort or merge for best performance, either explicitly or via use of pipes and standard I/O in Unix). You may use the Unix `sort` utility for the middle step, you can use parts of the code for merge sort provided on the course site, or you can implement your own I/O-efficient sorting procedure. But it has to be I/O-efficient, i.e., run fast even when main memory is much smaller than the data set. (It should be possible to limit the maximum amount of memory that your program will use through some parameter setting, and your program should work on data sets of multiple GB as long as it has at least a certain amount, say a few hundred MB, of memory.)

(3) **Parsing:** The data already has the HTML removed. To further parse the data, you may use a suitable parsing library. The output from this parser then has to be converted into the intermediate posting format and written to disk so that you can sort the postings afterwards.

(4) **Data Format:** The data that is provided is compressed. While you may uncompress the data before starting your program, it is preferable to uncompress during parsing, by loading parts of the compressed file into memory and then calling the right library function to uncompress it into another memory-based buffer.

(5) **Index File Format:** Your inverted index must be structured such that you can read a particular inverted list without reading the rest of the index, by looking up the start of the inverted list in the lexicon structure, and then seeking forward in the file. (Figure out how to perform random seeks on disk in your chosen programming language! Do not try to skip lines in an ascii file, but use byte offsets.) The index you build should consist of three files, one for the inverted index, one for the lexicon, and one for the page table – do not use a separate file for each inverted list! Try to find a reasonably space-efficient representation for the inverted lists based on var-byte compression or some other technique. For extra credit, you may try to keep postings in compressed format on disk even during the index building operation (i.e, the temporary files are compressed and then read in and written out in compressed form during the merge – of course, this means you would not be able to easily use Unix `sort` for this). Do not use generic file compression techniques such as `gzip` or `bzip2` to compress inverted lists!

(6) **DocIDs and Term IDs:** It is recommended to assign docIDs in the order in which the pages are parsed. You may either use term IDs in the intermediate posting format, or keep the terms in textual format. Of course, the final inverted lists should not have terms or term IDs inside each posting anymore.

(7) **Unicode:** Note that the input may contain text from many different languages, including text that uses non-ascii character sets. Read up on how to process data using unicode instead of basic ascii.

(8) **Future Assignments:** Make sure your code can be maintained and extended easily, as the next assignment will build further on this code base.

For this assignment, please hand in the following in electronic form: (a) your well-commented program source, and (b) a 4 to 5 page `readme.pdf` file explaining what your program can do, how to run the program, how it works internally, how long it takes on the provided data set, and how large the resulting index files are, etc. Explain your design decisions, what limitations it has, and what the major functions and modules are. There will be no demo for this assignment, but there will be one for the next assignment which builds on this. Each student should work on their assignment individually.