Polytechnic Institute of NYU February 17, 2013
Computer Science and Engineering
CS 6913, Spring 2013

Assignment #2 (due March 4)

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 probably all fairly slow, you will be provided a larger set of pages that you can use for this assignment; see the course web page. The data will be provided as a compressed tar file containing a few ten thousand up to a few million pages.

In the following, we will discuss the minimum requirements for your program, and also point out some opportunities for extra credit. 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 itself is provided in a fairly raw form with various errors (truncated web pages, 404 errors) that you need to be able to to deal with. Here are some hints and requirements:

(0) **Languages:** It is recommended that you use C/C^{++} or Java or Python 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 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 extra credit all disk-based structures should be in binary format (at least at the end, and preferably also for intermediate structures). Ideally, your program should have a compile flag that allows you to use ASCII format during debugging and binary format for performance.

(2) **Problem Decomposition:** You should implement this homework in two or three components, say one for generating intermediate postings from the files 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` or `merge` utilities for the middle step, or you may 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 the data set is much larger than the available main memory. (Ideally, 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 hundred MB, of memory.)

(3) **HTML Parsing:** To parse HTML pages, it is recommended that you use the C/C^{++}-based parser that is provided on the course page, though you can use other parsers or your own parser at your own risk. You can use the parser in languages other than C/C^{++} by calling it as a program or by wrapping it in an interface (using tools such as `SWIG`). 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. Do not spend much time trying to design your own parser – that is not the objective of this assignment!

(4) **Data Format:** Data is provided in a `tar` file that produces a directory of files. Each file is either a data file or an index file (with corresponding names), and files are compressed with `gzip`. You may uncompress the files before starting your program, though it is preferable to uncompress during parsing, ideally by loading one compressed file into memory and then calling the right library function to uncompress it into another memory-based buffer. Each file contains many pages.

(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. Your total index should consist of one or a few files, not one 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 use Unix `sort` or `merge` for this). Also, if you store the final inverted lists in compressed form, do not use generic compression techniques such as `gzip` or `bzip2` to do so.

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

(7) **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 2-3 page `readme.txt` 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, what limitations it has, and what the major functions and modules are. There will be no demo for this assignmnent, but there will be one for the next assignment which builds on this. As before, you may work in teams of two, but you need to let me know the names of the team members by February 22. No last minute switching!