NYU Tandon School of Engineering October 20, 2020
Computer Science and Engineering
CS 6913, Fall 2020

Assignment #3 (due November 6)

In this assignment, you are asked to write a program that uses the inverted index structure from the second homework to answer queries typed in by a user. Please use the same data set as for HW2. (If you were unable to complete HW2, but want to do HW3, please talk to me about what to do.) In the following, I will discuss the minimum requirements for your program, and also point out some opportunities for extra credit.

(0) **Languages:** It is recommended to use C/C^{++} or Java for this assignment. Or contact me if you prefer to use some other language. (If I have already given you permission to use another language for the second homework, then there is no need to ask me again.) But efficiency is very important. Even on the whole document set, your query processor should provide results in at most one or two seconds for typical queries, especially in the conjunctive case.

(1) **Query Execution:** It is recommended that your program should execute queries by using something similar to the simple interface presented in class based on operations *openList()*, *closeList()*, *nextGEQ()*, and *getFreq()* or *getPayload()* on inverted lists. (Recall that *nextGEQ(l, id)* returns the first posting in an already opened list l with docID at least id, when searching in a forward direction from the current position of the list pointer.) This way, issues such as file input and inverted list compression technique should be completely hidden from the higher-level query processor.

(2) **Ranked Queries:** Your query execution program should compute ranked queries according to the BM25 measure presented in class. Return the top 10 or 20 results according to the scoring function. You should implement conjunctive and disjunctive queries. You may experiment with additional ranking factors (e.g., context, proximity, Pagerank) for extra credit, as optional features. For each top result, you should also return its score according to the ranking function, as well as the frequency of each search term in the document. If you use additional factors such as Pagerank, also output those scores in a suitable way.

(3) **Interface:** Your program may read user queries via simple command line prompt, and you need to return the URL of each result, its BM25 score, and some snippet text with the context of the term occurrences in the pages. For extra credit, you could make your program web accessible, or work on smarter snippet generation algorithms, but this is not required.

(4) **Startup:** Upon startup, your query processor may read the complete lexicon and URL table data structures from disk into main memory, which might take some seconds or even a minute. However, you should not read the inverted index itself into memory! After the user inputs a query, your program should then perform seeks on disk in order to read **only those** inverted lists from disk that correspond to query words, and then compute the result. After returning the result, your program should wait for the next query to be input. You could optionally implement caching of inverted list data in main memory, but you should not assume that the entire index fits in main memory. Instead, upon starting the query processor, it should be possible to select the amount of memory that is used for index caching.

(5) **Index Compression:** Make sure that your inverted index is in binary format, and use some suitable form of index compression, such as Variable-Byte, Simple9, or another method. Do not use standard compressors such as gzip for index compression! It is recommended to use a block-wise compression scheme that allows skipping of blocks during query processing. When a query arrives, do not completely uncompress and temporarily store the complete inverted lists for the query terms − you should only decompress index data as needed inside the *nextGEQ()* and *getFreq()* operations.

For this assignment, please hand in the following in electronic form: (a) your well-commented program source, and (b) a 10-15 page paper (written in Word or Latex or similar) 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. Note: your paper should describe both your index structure and generation process from the previous homework, and the query processor from this homework. Revise your solution to the previous homework as needed. The paper should be readable to a generic computer scientist who knows what an inverted index is, but otherwise does not know much about search engines. So start very high-level! There will also be a demo for each student, which will be scheduled in the days after the deadline.