COS 435, Spring 2017 - Problem Set 3

Due 11:59 pm Wednesday March 8, 2017 by DropBox submission
~~***Due at 5:00 pm, Wednesday, March 8, 2017 if submitting handwritten work on paper.***~~

Please note we are no longer accepting work on paper. Please contact Mayank if you have no access to a scanner for handwritten work.

Collaboration and Reference Policy

You may discuss the general methods of solving the problems with other students in the class. However, each student must work out the details and write up his or her own solution to each problem independently. For each problem, list the students with whom you discussed general methods of solving the problem.

Some problems have been used in previous offerings of COS 435. You are NOT allowed to use any solutions posted for previous offerings of COS 435 or any solutions produced by anyone else for the assigned problems. You may use other reference materials; you must give citations to all reference materials that you use.

Lateness Policy

A late penalty will be applied, unless there are extraordinary circumstances and/or prior arrangements:

- Penalized 10% of the earned score if submitted by 10am Thursday (3/9/17).
- Penalized 25% of the earned score if submitted by 4:30 pm Friday (3/10/17).
- Penalized 50% if submitted later than 4:30 pm Friday (3/10/17).

Submission

Submit your solutions as a PDF file using the Computer Science Department DropBox submission system for COS435 at
https://dropbox.cs.princeton.edu/COS435_S2017/HW3 Name your file HW3.pdf. If you have not used this facility before, consult the instructions at
https://csguide.cs.princeton.edu/academic/csdropbox - student
Note that you are automatically enrolled in CS DropBox using the registrar's COS435 enrollment list.

Problem 1

Consider an inverted index containing, for each term, the postings list (i.e. the list of documents and occurrences within documents) for that term. The postings lists are accessed through a B+ tree with the terms serving as search keys. Each *leaf* of the B+ tree holds a sub-list of alphabetically consecutive terms, and, with each term, a *pointer to* the postings list for that term. (See the B^+ tree example in the slides posted for Mar. 1.)

Part a. Suppose there are 7 billion terms for a collection of 10 trillion documents of total size 100 petabytes. We would like each internal node of the B+ tree and each leaf of the B+ tree to fit in one 16-kilobyte (16*1024 byte) page of the file system. Recall that a B+ tree has a parameter **m** called the *order* of the tree, and each internal node of a B+ tree has between **m+1** and **2m+1** children (except the root, which has between 2 and **2m+1**). Assume that each term is represented using 40 bytes, and each pointer to a child in the tree or to a postings list is represented using 8 bytes. Find a value for the order **m** of the B+ tree so that one 16 kilobyte page can be assigned to each internal node and leaf, and so that an internal node will come as close as possible to filling its page without overflow its page when it has **2m+1** children. If you need to make additional assumptions, state what assumptions you are making.

Part b. For your **m** of Part a, estimate the height of the B+ tree for the inverted index of the collection described in Part a. (Giving a range of heights is fine.) Also estimate the amount of memory needed to store the tree, including leaves but not including the postings lists themselves. Assume that leaves contain between **m** and **2m** terms just as internal nodes do. Note that the height of a B+ tree is the length of the paths from root to leaves, e.g. a root with no children is height 0.

Problem 2

For this problem, assume that the scoring function for ranking documents with respect to a querys exact PageRank as well as term-based features, and possibly other features; the scoring details are not necessary for this problem.

In class we discussed sorting each postings list using PageRank (or some other global, a.k.a static, measure of the document value) with document ID as secondary sort key to give a total order. We did this to support a method of efficiently approximating “highest k” retrieval”. (This problem is called “inexact top *K* document retrieval” in our textbook *An Introduction to Information Retrieval*.) In class we discussed the general idea of such an algorithm. To be more concrete, consider the simple algorithm that merges the postings lists of the query terms and stops when **k** satisfying docs have been found. This algorithm assumes that since the postings lists are sorted by PageRank, the first **k** candidates found are a good enough approximation for the **k** highest ranking documents.

In this problem you are asked to consider the following alternative to sorting postings lists by PageRank: using a tiered index where each tier represents a range of PageRank values. The ranges for different tiers are disjoint. Postings lists within a tier are sorted on document ID, not on PageRank.

Part a. Describe an algorithm that uses the tiered index described above to ***quickly*** find a set of **k** documents that are good candidates for the top **k** documents satisfying a given multi-term query. Be clear about what documents are considered for ranking and how they are chosen. Note that the scoring function is using the exact PageRank, not simply which tier the documents is assigned.

Part b. Consider the simple algorithm using postings lists sorted by document PageRank (no tiers) that was described at the beginning of this problem. Will the results of your algorithm for Part a differ from those for the simple algorithm? Justify your answer. If you answer "yes", which algorithm do you think would give better results? Justify your answer.

Part c. Is one of the simple algorithm using postings lists sorted by PageRank and your algorithm of Part a computationally faster than the other? Justify your answer.

Problem 3

Part a. Outline the execution of the external sorting algorithm we discussed in class for the following (unrealistic) specifications:

- A buffer of 16 pages is available in RAM for use in the sorting.
- The list to be sorted is stored on disk in 8192 pages.

(Note that the size of a page in bytes is not relevant.)

As described in class, the external sorting algorithm consists of multiple passes through the entire list. In each pass either sortings of sub-lists or mergings of sub-lists occur. To describe the execution of the algorithm, state for each pass:

- what is being done
- the number and size in pages of the sub-lists being sorted or merged in the pass
- the number and size of the sub-lists resulting from the pass.

For numbers and sizes, ***be specific***. Do not use fractions for amounts that must be integers, like the number of lists. Give the exact number of pages in each sub-list, e.g. "this pass results in 24 sub-lists of 100 pages each and one sub-list of 4 pages" (made up numbers).

Part b. How many page reads from disk and page writes to disk are done in the entire execution?