COS 435, Spring 2011 - Problem Set 3

Due at 3:00pm, Wednesday, March 2, 2011.

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.

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:

- No penalty if in Prof. LaPaugh's office or inbox by 5pm Wednesday (3/2/11).
- Penalized 10% of the earned score if submitted by 11:59 pm Wed (3/2/11).
- Penalized 25% of the earned score if submitted by 5pm Friday (3/4/11).
- Penalized 50% if submitted later than 5pm Friday (3/4/11).

Problem 1 (2010 exam problem):
PageRank is usually applied to the graph of an entire collection of documents. However, we can also apply it to the subset of documents that satisfy a given query, just as HITS was used by its designers. Once we are focused on the links between documents that satisfy a query, we can also use the anchor text for those links. Assume we have a directed graph of nodes representing documents satisfying a query and edges indicating links between the documents. Furthermore, assume each edge is annotated to indicate whether the anchor text of the link contains at least one query term – a simple "yes" indicating the anchor text of the link contains at least one query term, or "no" indicating it does not.

The goal of this problem is to devise a method to more highly value the edges annotated "yes" in the computation of PageRank for the graph.

Part A
Propose a modification to the PageRank formulation that treats edges labeled "yes" and edges labeled "no" differently. Edges labeled "yes" should be favored as distributors of

PageRank. You may choose how to favor "yes" edges, but your modification must retain the random walk model. Be precise in your description. If you are changing the equation defining the PageRank of a node, you **must** give the new equation (or equations). Do you think your method converges in all cases? Why? (You **do not** need to show mathematically that your modified PageRank calculation converges.)

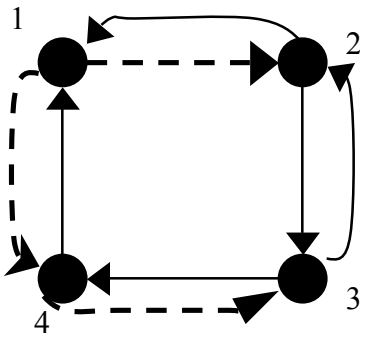
Part B

Apply your new PageRank to the small graph shown in Figure 1. Do only two iterations of an iterative computation of your new PageRank, and show the results of each iteration. Use $\alpha = 0.2$. Do you think it is converging? (Note: you may write a self-contained program in Java to do this calculation, but it is probably easier to do the calculation by hand. If you write a program, turn in the source code. You may **no**t use code written by someone else.)

Part C

What is the PageRank of the graph shown in Figure 1 under the original definition of PageRank?

Illustration of graph for Problem 1:



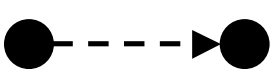
where  indicates anchor text containing at least one query term

Figure 1

Problem 2

Consider an inverted index containing, for each term, the posting list (i.e. the list of documents and occurrences within documents) for that term. The posting lists are accessed through a B+ tree with the terms serving as search keys. Each *leaf* of the B+ tree holds a sublist of alphabetically consecutive terms, and, with each term, a *pointer to* the posting list for that term. (See the "blow-up of B^+ tree example" posted on the Schedule and Assignments page under Feb. 21.)

Part a. Suppose there are 9 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 32 kilobyte 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 16 bytes, and each pointer to a child in the tree or to a posting list is represented using 8 bytes. Find a value for the order **m** of the B+ tree so that one 32 kilobyte page can be assigned to each internal node and leaf, and so that an internal node will fill, but not 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 posting lists themselves.

Part c. Estimate the aggregate size of the posting lists.

Problem 3

Consider a collection of technical reports. Each report has a title (including authors) and body of the report. For each report, each occurrence of each term and the position of that occurrence are recorded. It is also noted whether each occurrence is in the title or in the body of the report. Other attributes of term occurrences are also recorded.

Queries on this collection are sequences of terms. A report satisfies a query if it contains all the query terms (AND model). The ranking of reports that satisfy a query uses many attributes of the occurrences of the query terms, but is designed to guarantee that reports whose titles contain all the terms of the query score higher that reports whose titles contain only some or none of the query terms. Rankings of those satisfying reports whose titles contain all the terms of the query also are affected by occurrences of the query terms in the bodies of the reports so that the ranking can better distinguish between these reports.

Part a. Suppose the inverted index for this collection is organized in two parts: one inverted index recording only occurrences of terms in titles and a second inverted index recording only occurrences of terms in bodies of the reports. The postings for each term in each index are sorted by report ID. The occurrences of a term in the title of one report are sorted by position in the title of the report; the occurrences of a term in the body of one report are sorted by position in the body of the report.

For this index organization, describe **in detail** how the two-term query ***computer architecture*** would be processed. Describe exactly what information is used and how it is accessed. Assume you have pointers to the two postings lists for ***computer*** and the two postings lists for ***architecture*****;** assume the postings lists reside on disk**.** If you need to make an assumption about how information is stored, state your assumption**.**

Part b. Does this index organization speed up query processing compared to a standard inverted index containing one postings list per term? Justify your answer.