CS222: Homework 1. Due October 21.

Please work on problems on this assignment completely on your own. (You may of course ask the lecturer or the TF for assistance, but avoid working with other students.) You may use standard mathematical software (Maple, Matlab, etc.) and you may write programs to help you determine answers. You should do 4 of the 6 problems; if you do 5, we'll use your top 4 scores.

1. Determine Pagerank scores and Hub and Authority scores for the following graph (given by the transition matrix). State whatever assumptions you make in calculating the scores.

	a	b	c	d	e	f
a	0	0	1	1	1	0
b	0	0	1	0	1	0
c	0	0	0	1	1	0
d	0	1	1	0	0	0
e	0	0	0	0	0	1
f	1	0	0	0	0	0

2. Compare the performance of bzip, gzip, and compress (or comparable programs for your system– if you have to, download these utilities off the Web) on various file types and sizes. You should note encoding and decoding times as well; you may wish to experiment with different settings of parameters. (For example, gzip can try to compress more at the expense of speed.) (Write approximately 1 page.)

3. Suppose that I suggest the following alternative method for obtaining Hub and Authority scores, based on random walks. To obtain Authority scores, we consider the following random walk: from a page p_1, a step consists of first following back a random inlink from the current page to a page p_2, and then following forward a random outlink from this page to a page p_3. The step of the Markov chain takes us from p_1 to p_3. Similarly, Hub scores are computed as follows: from a page p_1, a step consists of first following forward a random outlink from the current page to a page p_2, and then following backward a random inlink from this page to a page p_3. The step of the Markov chain takes us from p_1 to p_3. We assume that these Markov chains are finite, irreducible, and aperiodic and hence have a unique stationary distribution.

Prove that, using this approach, the Hub score for a page is simply proportional to the number of outlinks and the Authority score is proportional to the number of inlinks.

4.

q w e r t y u i o p a s d f g h j k l z x c v b n m

e t a o n i h s r d l u m w c f g y p b v k q x j z

(The first is keyboard order; the second is English usage frequency.) Compute the footrule distance and the Kendall-tau distance between these two sequences. (Please give the UNSCALED values – integers, not a fraction between 0 and 1. You might note the scaling formulas in the paper are actually a bit off...)

Now (different subproblem) consider the following six ranking sequences, which would be returned by say a meta-search engine.

A B C D, D F E C, B E F A, B C D E, C F A B, A E F D.

Determine the scores for the pages A, B, C, D, E, and F under the chains MC3 and MC4 of the rank aggregation paper (either by finding the exact stationary distribution directly or by simulation).

5. The following approach is often called *reservoir sampling*. (It is useful, naturally, in streaming algorithms; you should think about why this might be.) Suppose that we have a sequence of items, passing by one at a time. We want to maintain a sample of one item that has the property that it is uniformly distributed over all the items that we have seen at each step. Moreover, we want to accomplish this without knowing the total number of items in advance or storing all of the items that we see.

Consider the following algorithm, which stores just one item in memory at all times. When the first item appears, it is stored in the memory. When the k-th item appears, it replaces the item in memory with probability $1/k$. Explain why this algorithm solves the problem.

Now suppose instead we want a sample of s items instead of just one, *without replacement*. That is, we don't want to get the same item multiple times in our sample. (If this wasn't an issue, we could get a sample of s items with replacement just by runnig s independent copies of the above.) Generalize the above process to that case. (Hint: start by taking the first s items and storing them as your sample. With what probability should each new item come into the sample?)

6. In this problem, you will experiment with the (multistage) filters of the Mice and Elephants/Count-Min sketches papers, both *with* and *without* conservative update. Your input stream will consist of the elements 1 through $9,100$. For $1 \leq i \leq 9$, elements $1000 \cdot (i-1)+1$ to $1000 \cdot i$ will appear i times in the stream. That is, elements 1 to 1000 will appear once in the stream; 1001 to 2000 will appear twice; and so on. Elements $9000+i$, for $1 \leq i \leq 100$, will appear i^2 times in the stream. For example, element 9100 will appear 10,000 times. (Each time an element appears in the stream, it has a count of 1 associated with it.)

You must consider the stream in 3 different orders:

1. all appearances of 1, followed by all appearances of 2, followed by all appearances of 3, etc. (forward order).
2. all appearances of 9100, followed by all appearances of 9109, followed by all appearances of 9108, etc. (reverse order)
3. all elements in a random order.

You should do 100 trials, both with conservative update and without. Use 4 independent hash tables and functions on each trial. Each hash table should have 500 counters. When hashing an item into a table, you can use an idealized pseudo-random hash function. This can be accomplished by using a pseudo-random function to determine where each element goes, and recording it in a table so you always put the element in the same place for the trial.

Over the 100 trials, 3 different orders, and with and without conservative update, record the following after all elements in the stream have passed:

1. How many elements are, according to the filter data structure, potentially responsible for 1% or more of the total load. Compare your result with the true answer.
2. What the presumed count is for element 9,100.

Present these results in a concise, organized manner.

Based on your results discuss the importance of order on filter performance, the importance of conservative update on filter performance, and your opinion overall on how the filter performed on this data stream.

For this problem, please turn in your code (in an appendix, where I hopefully won't have to read it).