University of Washington
Department of Computer Science and Engineering
CSEP 521, Winter 2021

February 3, 2021

Homework 5, Due Thursday, February 11, 2021

Problem 1 (10 points):

Give an algorithm with a small amount of memory that maintains a uniformly random item from a stream. After seeing items $a_1, a_2, \ldots, a_n$, the algorithm should have identified the item a_k with probability $\frac{1}{n}$.

Problem 2 (10 points):

Based on your solution to problem 1, propose an algorithm for an approximate solution to the Heavy Hitters problem, where the heavy hitters problem is, given an integer k, identify all items in a stream n elements occurring more than $\frac{n}{k}$ occurrences. The intent of this problem is for you develop an approximation algorithm that is not based on hashing. Explain why your algorithm might work. We are *not* expecting a mathematical analysis on this problem. You should identify the various parameters in your solution and explain how they will influence performance.

Problem 3 (10 points):

Lecture 8 introduced random graphs as an analysis tool for Cuckoo Hashing. For this problem you will determine the edge density required for the property "every vertex has degree at least one" to hold with high probability. For this problem, we will use the following model to generate random undirected graphs $G = (V, E)$ on n vertices with edge probability p: the vertex set is $V = \{1, 2, \ldots, n\}$ and the edge (i, j) is in E with probability p independent of other edges.

a) In this model, give a formula for the probability that a particular vertex has degree zero.

b) Let $p = \frac{2 \ln n}{n}$. What is the probability that a particular vertex has degree zero.

c) Let $p = \frac{2 \ln n}{n}$. Argue that the probability that the graph has any vertices of degree zero is $O(\frac{1}{n})$.

Note: We use $\ln n$ for the natural logarithm, so $e^{\ln k} = k$. A useful approximation in these types of calculations is $(1 - \frac{x}{n})^n \approx e^{-x}$.

Programming Problem 4 (20 points):

Experimentally evaluate the number of bits needed in a counting Bloom filter that supports deletion. The conventional wisdom is that four bit counters should suffice.

You are welcome to design an alternate set of experiments if you wish, but here is a suggested approach. Obviously if the load factor is too high, the counters are going to blow up, so you will need to have a target load factor β, the ratio of items to table size. Fill the table with inserts until you get to the load factor β, and then alternately delete an entry from the table, and insert a

new one. Deletes could be random or in a round robin, and the inserts will assume a random hash function. Repeat the inserts/delete operations for a large number of iterations. Does the counter size say below a fixed bound, or does it grow slowly over time. For parameters, I think the table can be fairly small, maybe 10,000 cells. For number of hash functions, maybe five. For number of iterations, you can choose a very large value on n, at least in the millions. Maybe more.
What value of β to use? For n items into a hash table of size m, using k hash functions, the probability that a cell is zero is $e^{kn/m}$, so if you want half the cells to be empty, you would take $\frac{n}{m} = .138$. Choosing a slightly larger value of β might be more interesting.