Homework 2

Algorithms for Big Data: CS498 ABD/ABG, Fall 2020
Due: Wednesday at 10pm CDT, 30th September 2020

Instructions and Policy:

- Each homework can be done in a group of size at most two. Only one homework needs to be submitted per group. However, we recommend that each of you think about the problems on your own first.
- Homework needs to be submitted in pdf format on Gradescope. See `https://courses.engr.illinois.edu/cs374/fa2018/hw-policies.html` for more detailed instructions on Gradescope submissions.
- Follow academic integrity policies as laid out in student code. You can consult sources but cite all of them including discussions with other classmates. Write in your own words. See the site mentioned in the preceding item for more detailed policies.

Problem 1. Hashing and limited independence. Let $h : [n] \to [m]$ be a random hash function chosen from a 3-wise independent family of hash functions. For a fixed item i let Y be the number of items $i' \neq i$ that collide with i under h.

- What is $\mathrm{E}[Y]$?
- What is $\mathrm{Var}[Y]$ as a function of m, n?
- Using Chebyshev, what is $\mathrm{P}[Y \geq a]$ where $a \geq 1$ is some integer. Express this as a function of a, m, n.

Problem 2. Hashing to the rescue. Suppose you want to estimate the number of distinct words in a long text document in English. The document is available in a streaming fashion (with one word at a time) but you have very limited memory and cannot store the entire document. You have access to a black-box streaming algorithm for distinct elements but the algorithm requires its input to be integers in $[n]$ for some large n. The size of the English vocabulary is n' where $n' \ll n$. However, the number of potential strings of 20 English characters (which is a reasonable upper bound on the word length) is $N \gg n$; thus we cannot use the black-box distinct elements algorithm by mapping each potential 20 character word to a distinct integer. Assume that you do not know how to solve distinct elements by yourself and hence need to use a reduction to the existing algorithm. How would you use hashing to address this problem? Briefly justify why you can obtain a good estimate with high probability.

Problem 3. Sampling distinct elements. We saw how to estimate the number of distinct elements from a stream. Now we consider the problem of sampling nearly uniformly from the set of distinct elements. If we had access to an ideal hash function $h : [n] \to [0, 1]$ then one can see that

the element which achieves the minimum hash value is uniformly distributed among the distinct elements. However, we do not have access to an ideal hash function. There is a notion of minwise independent hash functions which is directly relevant to this application; we will see their formal definition later in the course. For now we will see how to modify the algorithm we saw in lecture to obtain a near uniform sample from the distinct elements.

Consider the BJKST algorithm we saw in lecture that used a random hash function $h : [n] \to [n^3]$ from a pairwise independent hash family $\mathcal{H}$. Now assume that we instead use a 3-wise independent hash family. The algorithm stores $t = 1/\epsilon^3$ elements associated with the smallest t hash values seen and at the end of the algorithm outputs one of them uniformly at random. Our goal is to show that each distinct element is output with probability at least $(1 - 10\epsilon)/d$ and at most $(1 + \epsilon)/d$ (i.e., nearly uniform). (You may assume that ϵ is smaller than some constant like $1/2$ to make the calculations easier.)

To this end, let $b_1, b_2, \ldots, b_d$ be the distinct values in the stream. Assume $d > 1/\epsilon^3$ for otherwise we can store all of them and output a uniformly sampled element. Observe that an element b_i is among the t remaining elements if each of the following events all occur.

(a) $h(b_i) \leq \lceil (1 - \epsilon)tN/d \rceil$.

(b) The number of other elements b_j $(j \neq i)$ such that $h(b_j) < \lceil (1 - \epsilon)tN/d \rceil$ is at most $t - 1$.

(c) $h(b_j) \neq h(b_i)$ for all $j \neq i$.

Show that the above events all occur with probability close to t/d via the following steps.

1. Let Z be an indicator for $h(b_i) \leq \lceil (1 - \epsilon)tN/d \rceil$. What is $\mathrm{P}[Z = 1]$?
2. Conditioned on $Z = 1$, show that the probability that more than $t-1$ of the remaining items (b_j where $j \neq i$) has value $< \lceil (1 - \epsilon)tN/d \rceil$ is at most $(1 + \epsilon)\epsilon$.
3. Show that the probability of a hash collision with b_i is at most $\epsilon t/d$. *Hint: First show the probability is at most $O(1/n^2)$.*
4. Put the above together to show that b_i is one of the t selected elements with probability $\geq (1 - 10\epsilon)t/d$, hence output with probability $\geq \frac{1-10\epsilon}{d}$.
5. **Extra credit:** Show that the probability that b_i is output is at most $(1 + 10\epsilon)/d$ if you use a hash function with sufficiently large but constant independence.

Make note in particular of where you use the assumption that h is 3-wise independent.

Problem 4. Entropy estimation. In class, we saw how the AMS sampling procedure allows us to estimate the kth moment F^k in sublinear space for $k \geq 2$. Recall also that the AMS sampler still requires polynomial space because the variance of a single sample was polynomial. Here we will use AMS to estimate the *entropy* of a stream; in particular, we will show that we only need *logarithmic space* (modulo dependencies on ϵ and δ) to estimate the entropy.

Let $f \in \mathbb{N}_+^n$ be frequency counts over n elements, and for each i, let $p_i = \frac{f_i}{m}$ be the corresponding probability distribution. The *entropy* of p is the quantity $\Phi = \sum_i p_i \ln \frac{1}{p_i}$, where $0 \ln \frac{1}{0} = 0$.

Our high-level goal is to obtain an $(1 \pm \epsilon)$-multiplicative approximation to the entropy, but there is a technical issue because the entropy can be zero. We instead seek a $(1 \pm \epsilon)$-multiplicative approximation of $1 + \Phi$, which converts to a $(1 \pm 2\epsilon)$-multiplicative approximation of Φ if $\Phi \geq 1$ and a 2ϵ-additive approximation of Φ if $\Phi \leq 1$. You may assume m is larger than a fixed constant, say 42.

Let $g(\ell) = \frac{\ell}{em} \ln \frac{me}{\ell}$.

1. Show that $1 + \Phi = e \sum_{i \in [n]} g(f_i)$.
2. Show that $g(\ell) \geq g(\ell - 1)$ for $\ell \leq m$.
3. Show that for $\ell \leq m$, $g(\ell) - g(\ell - 1) \leq \frac{1 + \ln m}{me}$
4. Let Y be the AMS sample based estimator for the quantity $\frac{1}{e}(1 + \Phi) = \sum_i g(f_i)$. We know from class that $\mathrm{E}[Y] = \frac{1}{e}(1 + \Phi)$. Show that

$$\mathrm{Var}[Y] \leq c_1(1 + \ln m)(1 + \Phi) = ec_1(1 + \ln m)\,\mathrm{E}[Y]$$

   for some constant $c_1 > 0$.
5. Use averaging and median trick to obtain a $(1 \pm \epsilon)$-approximation of $1 + \Phi$ with probability $\geq 1 - \delta$. What is the total space usage?

Problem 5. Join size estimation. Recall that for two relations (i.e., tables in a database) $r(A, B)$ and $s(A, C)$, with a common attribute (i.e., column) A, we define the join $r \bowtie s$ to be a relation consisting of all tuples (a, b, c) such that $(a, b) \in r$ and $(a, c) \in s$. Therefore, if $f_{r,j}$ and $f_{s,j}$ denote the frequencies of j in the first columns (i.e., A-columns) of r and s, respectively, and j can take values in $[n]$, then the size of the join is $\sum_{j=1}^{n} f_{r,j} f_{s,j}$. As we noted in lecture if $r = s$ then $\sum_{j=1}^{n} f_{r,j} f_{s,j} = \sum_{j=1}^{n} f_{r,j}^2$ for which we can use F_2 estimation algorithm (this is the self join case). Generalize this estimation algorithm when r and s are not necessarily the same relation. More formally design a sketch from a scan of a relation in one streaming pass such that, based on the sketches of two different relations, we can estimate the size of their join. Explain how to compute the estimate. Show, via an example class of instances, that the variance of this estimator can be very large compared to the actual value.