CS 170 HW 14 (Optional)

Due **2019-12-11, at 10:00 pm**

1 Study Group

List the names and SIDs of the members in your study group.

2 One-to-One Functions

Suppose that $\mathcal{H}$ is a pairwise independent hash family that maps the elements $U = \{1, \dots, n\}$ to $1, \dots, n^3$. Show that the probability that a randomly chosen hash function h from $\mathcal{H}$ is one-to-one is at least $1 - 1/n$.

Solution: Consider the probability that a random function drawn at uniform from $\mathcal{H}$ is one to one:

$$\begin{aligned}
\Pr_{h\in\mathcal{H}}(h \text{ is one-to-one}) &= \Pr_{h\in\mathcal{H}}(h(x_1) \neq h(x_2) \neq \cdots \neq h(x_n)) \\
&= 1 - \Pr_{h\in\mathcal{H}}(\bigcup_{i\neq j} h(x_i) = h(x_j)) \\
&\overset{\zeta_1}{\geq} 1 - n(n-1)\left[\Pr_{h\in\mathcal{H}}(h(x_i) = h(x_j))\right] \\
&\overset{\zeta_2}{\geq} 1 - n(n-1)\left[\frac{n^3}{n^6}\right] \\
&\geq 1 - \frac{1}{n},
\end{aligned}$$

where ζ_1 follows from a union bound over $n(n-1)$ pairs of (i, j) and ζ_2 follows from the definition of pairwise independent hash family for any x_i, x_j.

3 Universal Hashing

Let $[m]$ denote the set $\{0, 1, ..., m-1\}$. For each of the following families of hash functions, determine whether or not it is pairwise-independent. If it is universal, determine how many random bits are needed to choose a function from the family.

(a) $H = \{h_{a_1,a_2} : a_1, a_2 \in [m]\}$, where m is a fixed prime and

$$h_{a_1,a_2}(x_1, x_2) = a_1x_1 + a_2x_2 \mod m$$

Notice that each of these functions has signature $h_{a_1,a_2} : [m]^2 \to [m]$, that is, it maps a pair of integers in $[m]$ to a single integer in $[m]$.

(b) H is as before, except that now $m = 2^k$ is some fixed power of 2.

(c) H is the set of all functions $f : [m] \to [m-1]$.

Solution:

(a) The hash function is universal. The universality proof is the same as the one in the textbook (only now we have a 2-universal family instead of 4-universal). To reiterate, assume we are given two distinct pairs of integers $x = (x_1, x_2)$ and $y = (y_1, y_2)$. Without loss of generality, let's assume that $x_1 \neq y_1$. If we chose values a_1 and a_2 that hash x and y to the same value, then $a_1x_1 + a_2x_2 \equiv a_1y_1 + a_2y_2 \mod m$. We can rewrite this as $a_1(x_1 - y_1) \equiv a_2(y_2 - x_2) \mod m$. Let $c \equiv a_2(y_2 - x_2) \mod m$. Since m is prime and $x_1 \neq y_1$, $(x_1 - y_1)$ must have a unique inverse. So $a_1(x_1 - y_1) \equiv a_2(y_2 - x_2) \mod m$ if and only if $a_1 \equiv c(x_1 - y_1)^{-1} \mod m$, which will only happen with probability $1/m$.

We need to randomly pick two integers in the range $[0, \ldots, m-1]$, so we need $2 \log m$ random bits.

(b) This family is not universal. Consider the following inputs: $(x_1, x_2) = (0, 2^{k-1})$ and $(y_1, y_2) = (2^{k-1}, 0)$. We then have $h_{\alpha_1,\alpha_2}(x_1, x_2) = 2^{k-1}\alpha_2 \mod 2^k$ and $h_{\alpha_1,\alpha_2}(y_1, y_2) = 2^{k-1}\alpha_1 \mod 2^k$. Now notice that if α_2 is even (i.e. with probability 1/2) then $h_{\alpha_1,\alpha_2}(x_1, x_2) = 0 \mod 2^k$ otherwise (if α_2 is odd) $h_{\alpha_1,\alpha_2}(x_1, x_2) = 2^{k-1} \mod 2^k$; likewise for α_1. So we get that $h_{\alpha_1,\alpha_2}(x_1, x_2) = h_{\alpha_1,\alpha_2}(y_1, y_2)$ with probability $1/2 > 1/2^k$, so the family is not universal.

(c) This family is universal. To see that, fix $x, y \in \{0, 1, \ldots, m-1\}$ with $x \neq y$. Now we need to figure out the following: how many (out of the $(m-1)^m$ in total) functions $f : [m] \to [m-1]$ will collide on x and y, i.e. $f(x) = f(y) = k$, for some fixed $k \in [m-1]$. Well, there are $(m-1)^{m-2}$ different functions $f : [m] \to [m-1]$ that have the property $f(x) = f(y) = k$ (because I just fixed the output of 2 inputs to some fixed $k \in [m-1]$ and allow the output of f for all other inputs to range over all $m-1$ possible values). Finally, ranging over all $m-1$ values of k, we get that there are $(m-1)^{m-1}$ functions $f : [m] \to [m-1]$ with the property $f(x) = f(y)$. So the probability of picking one such f is exactly $\frac{(m-1)^{m-1}}{(m-1)^m} = \frac{1}{m-1}$.

How many bits do we need in this case? Well, there is no succinct representation in this case, so we need to write down the whole family of functions explicitly and then pick one of the $(m-1)^m$ functions of the family. To do that we can imagine indexing all functions with integers $1, \ldots, (m-1)^m$ and randomly picking one such integer $k \in \{1, \ldots, (m-1^m)\}$; this obviously requires $\log(m-1)^m = m\log(m-1)$ bits.

4 Count-Median-Sketch

Consider the *Count-Min-Sketch* algorithm from class for determining heavy hitters in a stream. Suppose that we change the algorithm to consider $\text{median}_{i=1,\ldots,l} M[i, h_i(x)]$ instead of the min function.

(a) Give an argument that the estimator for heavy hitters is still correct.

Note: We are not looking for a rigorous proof here. Just an argument is fine.

(b) If we allow the stream to both insert and delete elements, show that the median based algorithm works to find heavy hitters. Each item x in the stream now comes with a an element $\Delta \in \{+1, -1\}$ indicating whether it is an addition operation or deletion. Note that the number of deletions could be more than the number of additions.

Note: We are not looking for a rigorous proof here. Just an argument is fine.

(c) Would *Count-Min-Sketch* work for the setup above?

Solution:

(a) As pointed out in the notes, $\mathbb{E}(M[i, h_i(a)]) \leq f_a + \frac{n}{B}$, where n is the length of the stream and B is the size of the dictionary to which the hash functions h_i map. Using Markov inequality, we get that:

$$\Pr\left[M[i, h_i(a)] \geq f_a + \frac{3n}{B}\right] \leq \frac{1}{3}. \tag{1}$$

For $i = \{1, \ldots, l\}$, let Z_i be a Bernoulli variable such that $Z_i = 1$ if $M[i, h_i(a)] \geq f_a + \frac{3n}{B}$. For the median estimator to work, we need to ensure that $\Pr[\sum_i Z_i > l/2]$ is small. Using a Hoeffding bound, we get:

$$\begin{aligned}
\Pr\left(\sum_i Z_i > l/2]\right) &= \Pr\left(\frac{1}{l}\sum_i Z_i - \mathbb{E}(Z_i) > \frac{1}{2} - \mathbb{E}(Z_i)\right) \\
&\leq \Pr\left(\frac{1}{l}\sum_i Z_i - \mathbb{E}(Z_i) > \frac{1}{2} - \frac{1}{3}\right) \\
&\leq \Pr\left(\frac{1}{l}\sum_i Z_i - \mathbb{E}(Z_i) > \frac{1}{6}\right) \leq \exp\left(-\frac{l}{18}\right),
\end{aligned}$$

where the firs inequality follows from Equation (**??**). Setting $B = 30$ and $l = 36 \cdot \ln n$ gives us that with probability atleast $1 - 1/n^2$, the estimate $\text{median}_{i=1,\ldots,l} M[i, h_i(x)]$ is within f_a and $f_a + 0.1n$.

(b) In this deletion setup, the *Count-Median-Sketch* algorithm would still perform well. The idea is that for any heavy hitter item a, for the median algorithm to fail, more than half of the l hash functions need to have collisions with other items for which the number of deletions is more than the number of insertions. The hash functions are designed to have a low probability of collision and hence, having more than $l/2$ collisions is a very low-probability event.

(c) No, there is a high chance for *Count-Min-Sketch* to fail for the setup above. Consider any item a which is a heavy hitter. If the minimum entry of the hash table corresponding to this item clashes with another item b and if the number of deletions of b are more than insertions by a margin of $0.2n$, the algorithm will not output a as a heavy hitter.

5 Document Comparison with Streams

You are given a document A and then a document B, both as streams of words. Find a streaming algorithm that returns the degree of similarity between the words in the documents,

given by $\frac{|I|}{|U|}$, where I is the set of words that occur in both A and B, and U is the set of words that occur in at least one of A and B.
Clearly explain your algorithm and briefly justify its correctness and memory usage (at most $\log(|A| + |B|)$. Can we achieve accuracy to an arbitrary degree of precision? That is, given any $\epsilon > 0$ can we guarantee that the solution will always be within a factor of $1 \pm \epsilon$ with high probability?

Solution: Simply use the number of distinct elements streaming algorithm we saw in class, on the streams of A, B, and $C := A \cup B$ (for this third stream, process words in A and words in B). Let $|A|, |B|$, and $|C|$ be the output of the algorithm on the corresponding set, our estimate for the number of distinct words in it. Then, $|U| = |C|$, and $|I| = |A| + |B| - |U|$. Thus our estimate for $|I|/|U| = (|A| + |B| - |C|)/|C|$.
Memory usage is logarithmic in the length of the documents, as we're using a constant number (three) copies of the streaming algorithm shown in class, which used logarithmic memory. Correctness flows from the correctness of the underlying streaming algorithm for distinct elements, combined with the elementary axiom in set theory that $|A \cap B| = |A| + |B| - |A \cup B|$.
We can achieve accuracy within an arbitrary factor, because the accuracy of the similarity estimate depends on the accuracy of the underlying number of distinct elements algorithm, which we saw in class gives a result which is with high probability a fraction $(1 \pm \epsilon)$ of the true value.
The full mathematical details are beyond our scope, but we can do a rough analysis to gauge our algorithm's accuracy: the numerator is the sum of three outputs of the streaming algorithm, so is with high probability $1 \pm 3\epsilon$, and the denominator is within $1 \pm \epsilon$, of their true values, so the overall fraction is with high probability within $(1 \pm 3\epsilon)/(1 \mp 1\epsilon)$. The worst case in this range would be $(1 + 3\epsilon)/(1 - \epsilon)$, (or swap the + and - signs), which simplifies to $\frac{(1+3\epsilon)(1+\epsilon)}{(1-\epsilon)(1+\epsilon)} = \frac{1+4\epsilon+3\epsilon^2}{1-\epsilon^2} \approx 1 + 4\epsilon$, because ϵ^2 is negligible. In the other case, we obtain $1 - 4\epsilon$, so we conclude the overall accuracy is likely to be within $1 \pm 4\epsilon$. This calculation is not exact, but suffices to argue that the accuracy of the overall algorithm is a function of the accuracy of the underlying number of distinct elements algorithm it used, which can be set arbitrarily low.

6 Streaming Integers

In this problem, we assume we are given an infinite stream of integers x_1,x_2, ..., and have to perform some computation after each new integer is given. Since we may see many integers, we want to limit the amount of memory we have to use in total. For all of the parts below, give a brief description of your algorithm and a brief justification of its correctness.

(a) Show that using only a single bit of memory, we can compute whether the sum of all integers seen so far is even or odd.

(b) Show that we can compute whether the sum of all integers seen so far is divisible by some fixed integer N using $O(\log N)$ bits of memory.

(c) Assume N is prime. Give an algorithm to check if N divides the product of all integers seen so far, using as few bits of memory as possible.

(d) Now let N be an arbitrary integer, and suppose we are given its prime factorization: $N = p_1^{k_1} p_2^{k_2} \dots p_r^{k_r}$. Give an algorithm to check whether N divides the product of all integers seen so far, using as few bits of memory as possible. Write down the number of bits your algorithm uses in terms of $k_1, \dots, k_r$.

Solution:

(a) We set our single bit to 1 if and only if the sum of all integers seen so far is odd. This is sufficient since we don't need to store any other information about the integers we've seen so far.

(b) Set $y_0 = 0$. After each new integer x_i, we set $y_i = y_{i-1} + x_i \mod N$. The sum of all seen integers at step i is divisible by N if and only if $y_i \equiv 0 \mod N$. Since each y_i is between 0 and $N-1$, it only takes $\log N$ bits to represent y_i.

(c) We can do this with a single bit b. Initially set $b = 0$. Since N is prime, N can only divide the product of all x_is if there is a specific i such that N divides x_i. After each new x_i, check if N divides x_i. If it does, set $b = 1$. b will equal 1 if and only if N divides the product of all seen integers.

(d) We can do this with $\lceil \log_2(k_1) \rceil + \lceil \log_2(k_2) \rceil + \cdots + \lceil \log_2(k_r) \rceil$ bits. For each i between 1 and r, we track the largest value $t_i \leq k_i$ such that p^{t_i} divides the product of all seen numbers. We start with $t_i = 0$ for all i. When a new number m is seen, we find the largest t'_i such that $p^{t'_i}$ divides m and set $t_i = \min\{t'_i + t_i, k_i\}$ We stop once $t_i = k_i$ for all i as this implies that N divides the product of all seen numbers.