CS 598CSC: Algorithms for Big Data	Lecture date: August 28, 2014
Instructor: Chandra Chekuri	Scribe: Chandra Chekuri

1 Estimating Frequency Moments in Streams

A significant fraction of streaming literature is on the problem of estimating frequency moments. Let $\sigma = a_1, a_2, \ldots, a_m$ be a stream of numbers where for each i, a_i is an intger between 1 and n. We will try to stick to the notation of using m for the length of the stream and n for range of the integers[1]. Let f_i be the number of occurences (or frequency) of integer i in the stream. We let $\mathbf{f} = (f_1, f_2, \ldots, f_n)$ be the frequency vector for a given stream σ. For $k \geq 0$, $F_k(\sigma)$ is defined to be the k'th frequency moment of σ:

$$F_k = \sum_i f_i^k.$$

We will discuss several algorithms to estimate F_k for various values of k. For instance F_0 is simply the number of distinct elements in σ. Note that $F_1 = \sum_i f_i = m$, the length of the stream. A k increases to ∞ F_k will concentrate on the most frequent element and we can thing of F_∞ as finding the most frequent element.

Definition 1 *Let $\mathcal{A})(\sigma)$ be the real-valued output of a randomized streaming algorithm on stream σ. We say that $\mathcal{A}$ provides an (α, β)-approximation for a real-valued function g if*

$$\Pr\left[|\frac{\mathcal{A}(\sigma)}{g(\sigma)} - 1| > \alpha\right] \leq \beta$$

for all σ.

Our ideal goal is to obtain a (ϵ, δ)-approximation for any given $\epsilon, \delta \in (0, 1)$.

2 Background on Hashing

Hashing techniques play a fundamental role in streaming, in particular for estimating frequency moments. We will briefly review hashing from a theoretical point of view and in particular k-universal hashing.

A hash function maps a finite universe $\mathcal{U}$ to some range $\mathcal{R}$. Typically the range is the set of integers $[0..L-1]$ for some finite L (here L is the number of buckets in the hash table). Sometimes it is convenient to consider, for the sake of developing intuition, hash functions that maps $\mathcal{U}$ to the continuous interval $[0, 1]$. We will, in general, be working with a family of hash functions $\mathcal{H}$ and h will be drawn from $\mathcal{H}$ uniformly at random; the analyis of the algorithm will be based on properties of $\mathcal{H}$. We would like $\mathcal{H}$ to have two important and contradictory properties:

- a random function from $\mathcal{H}$ should behave like a completely random function from $\mathcal{U}$ to the range.
- $\mathcal{H}$ should have nice computational properties:

[1] Many streaming papers flip the notation and use n to denote the length of the stream and m to denote the range of the integers in the stream

- a uniformly random function from $\mathcal{H}$ should be easy to sample
- any function $h \in \mathcal{H}$ should have small representation size so that it can be stored compactly
- it should be efficient to evaluate h

Definition 2 *A collection of random variables $X_1, X_2, \ldots, X_n$ are k-wise independent if the variables $X_{i_1}, X_{i_2}, \ldots, X_{i_k}$ are independent for any set of distinct indices $i_1, i_2, \ldots, i_k$.*

Take three random variables $\{X_1, X_2, X_3\}$ where X_1, X_2 are independent $\{0,1\}$ random variables and $X_3 = X_1 \oplus X_2$. It is easy to check that the three variables are pairwise independent although they are not all independent.

Following the work of Carter and Wegman [3], the class of k-universal hash families, and in particular for $k = 2$, provide an excellent tradeoff. $\mathcal{H}$ is strongly 2-universal if the following properties hold for a random function h picked from $\mathcal{H}$: (i) for every $x \in \mathcal{U}$, $h(x)$ (which is a random variable) is uniformly distributed over the range and (ii) for every distinct pair $x, y \in \mathcal{U}$, $h(x)$ and $h(y)$ are independent. 2-universal hash families are also called pairwise independent hash families. A weakly 2-universal family satisfies the property that that $\Pr[h(x) = h(y)] = 1/L$ for any distinct x, y. We state an important observation about pairwise independent random variables.

Lemma 1 *Let $Y = \sum_{i=1}^{h} X_i$ where $X_1, X_2, \ldots, X_h$ are pairwise independent. Then*

$$\mathbf{Var}[Y] = \sum_{i=1}^{h} \mathbf{Var}[X_i].$$

Moreover if X_i are binary/indicator random variables then

$$\mathbf{Var}[Y] \leq \sum_i \mathbf{E}[X_i^2] = \sum_i \mathbf{E}[X_i] = \mathbf{E}[Y].$$

There is a simple and nice construction of pairwise independent hash functions. Let p be a prime number such that $p \geq |\mathcal{U}|$. Recall that $Z_p = \{0, 1, \ldots, p-1\}$ forms a field under the standard addition and multiplication modulo p. For each $a, b \in [p]$ we can define a hash function $h_{a,b}$ where $h_{a,b}(x) = ax + b \mod p$. Let $\mathcal{H} = \{h_{a,b} \mid a, b \in [p]\}$. We can see that we only need to store two numbers a, b of $\Theta(\log p)$ bits to implicitly store $h_{a,b}$ and evaluation of $h_{a,b}(x)$ takes one addition and one multiplication of $\log p$ bit numbers. Moreover, samply a random hash function from $\mathcal{H}$ requires sampling a, b which is also easy. We claim that $\mathcal{H}$ is a pairwise independent family. You can verify this by the observation that for distinct x, y and any i, j the two equations $ax + b = i$ and $ay + b = j$ have a unique a, b them simultaneously. Note that if $a = 0$ the hash function is pretty useless; all elements get mapped to b. Nevertheless, for $\mathcal{H}$ to be pairwise independent one needs to include those hash functions but the probability that $a = 0$ is $1/p$ while there are p^2 functions in $\mathcal{H}$. If one only wants a weakly universal hash family we can pick a from $[1..(p-1)]$. The range of the hash function is $[p]$. To restrict the range to L we let $h'_{a,b}(x) = (ax + b \mod p) \mod L$.

More generally we will say that $\mathcal{H}$ is k-universal if every element is uniformly distributed in the range and for any k elements $x_1, \ldots, x_k$ the random variabels $h(x_1), \ldots, h(x_k)$ are independent. Assuming $\mathcal{U}$ is the set of integers $[0..|\mathcal{U}|]$, for any fixed k there exist constructions for k-universal hash families such that every hash function h in the family can be stored using $O(k \log |\mathcal{U}|)$ bits (essentially k numbers) and h can be evaluated using $O(k)$ arithmetic operations on $\log |\mathcal{U}|$ bit numbers. We will ignore specific details of the implementations and refer the reader to the considerable literature on hashing for further details.

3 Estimating Number of Distinct Elements

A lower bound on exact counting deterministic algorithms: We argue that any deterministic streaming algorithm that counts the number of distinct elements exactly needs $\Omega(n)$ bits. To see this, suppose there is an algorithm $\mathcal{A}$ that uses strictly less than n bits. Consider the $h = 2^n$ different streams σ_S where $S \subseteq [n]$; σ_S consists of the elements of S in some arbitrary order. Since $\mathcal{A}$ uses $n-1$ bits or less, there must be two distinct sets S_1, S_2 such that the state of $\mathcal{A}$ at the end of $\sigma_{S_1}, \sigma_{S_2}$ is identical. Since S_1, S_2 are distinct there is an element i in $S_1 \setminus S_2$ or $S_2 \setminus S_1$; wlog it is the former. Then it is easy to see the $\mathcal{A}$ cannot give the right count for at least one of the two streams, $< \sigma_{S_1}, i >$, $< \sigma_{S_2}, i >$.

The basic hashing idea: We now discuss a simple high-level idea for estimating the number of distinct elements in the stream. Suppose h is an idealized random hash function that maps $[1..n]$ to the interval $[0, 1]$. Suppose there are d distinct elements in the stream $\sigma = a_1, a_2, \ldots, a_m$. If h behaves like a random function then the set $\{h(a_1), \ldots, h(a_m)\}$ will behave like a collection of d independent uniformly distributed random variables in $[0, 1]$. Let $\theta = \min\{h(a_1), \ldots, h(a_m)\}$; the expectation of θ is $\frac{1}{d+1}$ and hence $1/\theta$ is good estimator. In the stream setting we can compute θ by hashing each incoming value and keeping track of the minimum. We only need to have one number in memory. Although simple, the algorithm assumes idealized hash functions and we only have an unbiased estimator. To convert the idea to an implementable algorithm with proper guarantees requires work. There are several papers on this problem and we will now discuss some of the approaches.

3.1 The AMS algorithm

Here we describe an algorithm with better parameters but it only gives a constant factor approximation. This is due to Alon, Matias and Szegedy in their famous paper [1] on estimating frequency moments. We need some notation. For an integer $t > 0$ let zeros(t) denote the number of zeros that the binary representation of t ends in; equivalenty

$$\text{zeros}(t) = \max\{i :| 2^i \text{ divides } t\}.$$

AMS-DistinctElements:

$\mathcal{H}$ is a 2-universal hash family from $[n]$ to $[n]$
choose h at random from $\mathcal{H}$
$z \leftarrow 0$
While (stream is not empty) do
  a_i is current item
  $z \leftarrow \max\{z, \text{zeros}(h(a_i))\}$
endWhile
Output $2^{z+\frac{1}{2}}$

First, we note that the space and time per element are $O(\log n)$. We now analyze the quality of the approximation provided by the output. Recall that $h(a_j)$ is uniformly distributed in $[n]$. We will assume for simplicity that n is a power of 2.

Let d to denote the number of distinct elements in the stream and let them be $b_1, b_2, \ldots, b_d$. For a given r let $X_{r,j}$ be the indicator random variable that is 1 if zeros$(h(b_j)) \geq r$. Let $Y_r = \sum_j X_{r,j}$. That is, Y_r is the number of distinct elements whose hash values have atleast r zeros.

Since $h(b_j)$ is uniformaly distribute in $[n]$,

$$\mathbf{E}[X_{r,j}] = \Pr[\text{zeros}(h(b_j)) \geq r] = \frac{(n/2^r)}{n} \geq \frac{1}{2^r}.$$

Therefore

$$\mathbf{E}[Y_r] = \sum_j \mathbf{E}[X_{r,j}] = \frac{d}{2^r}.$$

Thus we have $\mathbf{E}[Y_{\log d}] = 1$ (assuming d is a power of 2).

Now we compute the variance of Y_r. Note that the variables $X_{r,j}$ and $X_{r,j'}$ are pairwise independent since $\mathcal{H}$ is 2-universal. Hence

$$\mathbf{Var}[Y_r] = \sum_j \mathbf{Var}[X_{r,j}] \leq \sum_j \mathbf{E}[X_{r,j}^2] = \sum_j \mathbf{E}[X_{r,j}] = \frac{d}{2^r}.$$

Using Markov's inequality

$$\Pr[Y_r > 0] = \Pr[Y_r \geq 1] \leq \mathbf{E}[Y_r] \leq \frac{d}{2^r}.$$

Using Chebyshev's inequality

$$\Pr[Y_r = 0] = \Pr[|Y_r - \mathbf{E}[Y_r]| \geq \frac{d}{2^r}] \leq \frac{\mathbf{Var}[Y_r]}{(d/2^r)^2} \leq \frac{2^r}{d}.$$

Let z' be the value of z at the end of the stream and let $d' = 2^{z'+\frac{1}{2}}$ be the estimate for d output by the algorithm. We claim that d' cannot be too large compared to d with constant probability. Let a be the smallest integer such that $2^{a+\frac{1}{2}} \geq 3d$.

$$\Pr[d' \geq 3d] = \Pr[Y_a > 0] \leq \frac{d}{2^a} \leq \frac{\sqrt{2}}{3}.$$

Now we claim that d' is not too small compared to d with constant probability. For this let b the largest integer such that $2^{b+\frac{1}{2}} \leq d/3$. Then,

$$\Pr[d' \leq d/3] = \Pr[Y_{b+1} = 0] \leq \frac{2^{b+1}}{d} \leq \frac{\sqrt{2}}{3}.$$

Thus, the algorithm provides $(1/3, \sqrt{2}/3 \simeq 0.4714)$-approximation to the number of distinct elements. Using the median trick we can make the probability of success be at least $(1-\delta)$ to obtain a $(1/3, \delta)$-approximation by running $O(\log \frac{1}{\delta})$-parallel and independent copies of the algorithm. The time and space will be $O(\log \frac{1}{\delta} \log n)$.

3.2 A $(1-\epsilon)$-approximation in $O(\frac{1}{\epsilon^2} \log n)$ space

Bar-Yossef et al. [2] described three algorithms for distinct elements, that last of which gives a bound $\tilde{O}(\epsilon^2 + \log n))$ space and amortized time per element $O(\log n + \log \frac{1}{\epsilon})$; the notation $\tilde{O})$ suppresses dependence on $\log \log n$ and $\log 1/\epsilon$. Here we describe their first algorithm that gives gives an (ϵ, δ_0)-approximation in space $O(\frac{1}{\epsilon^2} \log n)$ and $O(\log 1/\epsilon \log n)$ time per update; via the median trick, with an additional $\log 1/\delta$ factor, we can obtain a (ϵ, δ)-approximation.

The algorithm is based on the Flajolet-Martin idea of hashing to $[0, 1]$ and taking the minimum but with a small and important technical tweak. Let $t = \frac{c}{\epsilon^2}$ for some constant c to be fixed later.

BJKST-DistinctElements:

$\mathcal{H}$ is a 2-universal hash family from $[n]$ to $[N = n^3]$
choose h at random from $\mathcal{H}$
$t \leftarrow \frac{c}{\epsilon^2}$
While (stream is not empty) do
  a_i is current item
  Update the smallest t hash values seen so far with $h(a_i)$
endWhile
Let v be the t'th smallest value seen in the hast values.
Output tN/v.

We observe that the algorithm can be implemented by keeping track of t hash values each of which take $O(\log n)$ space and hence the space usage is $O(t \log n) = O(\frac{1}{\epsilon^2} \log n)$. The t values can be stored in a binary search tree so that when a new item is processed we can update the data structure in $O(\log t)$ searches which takes total time $O(\log \frac{1}{\epsilon} \log n)$.

The intution of the algorithm is as follows. As before let d be the number of distinct elements and let them be $b_1, \ldots, b_d$. The hash values $h(b_1), \ldots, h(b_d)$ are uniformly distributed in $[0, 1]$. For any fixed t we expect about t/d hash values to fall in the interval $[0, t/d]$ and the t'th smallest hash value v should be around t/d and this justifies the estimator for d being t/v. Now we formally analyse the properties of this estimator.

We chose the hash family to map $[n]$ to $[n^3]$ and therefore with probability at least $(1 - 1/n)$ the random hash function h is injective over $[n]$. We will assume this is indeed the case. Moreover we will assume that $n \geq \sqrt{\epsilon/24}$ which implies in particular that $\epsilon t/(4d) \geq 1/N$. Let d' the estimate returned by the algorithm.

Lemma 2 $\Pr[d' < (1 - \epsilon)d] \leq 1/6]$.

Proof: The values $h(b_1), \ldots, h(b_d)$ are uniformly distributed in $1..N$. If $d' < (1 - \epsilon)d$ it implies that $v > \frac{tN}{(1-\epsilon)d}$; that is less than t values fell in the interval $[1, \frac{tN}{(1-\epsilon)d}]$. Let X_i be the indicator random variable for the event that $h(b_i) \leq (1 + \epsilon)tN/d$ and let $Y = \sum_{i=}^{d} X_i$.

Since $h(b_i)$ is distributed uniformly in $[1..N]$, taking rounding errors into account, we have $(1 + \epsilon/2)t/d \leq \mathbf{E}[X_i] \leq (1 + 3\epsilon/2)t/d$ and hence $\mathbf{E}[Y] \geq t(1 + \epsilon/2)$. We have $\mathbf{Var}[Y] \leq \mathbf{E}[Y] \leq t(1 + 3\epsilon/2)$ (due to pairwise independence, Lemma 1)). We have $d' < (1 - \epsilon)d$ only if $Y < t$ and by Chebyshev,

$$\Pr[Y < t] \leq \Pr[|Y - \mathbf{E}[Y]| \geq \epsilon t/2] \leq \frac{4\mathbf{Var}[Y]}{\epsilon^2 t^2} \leq 12(\epsilon^2 t^2) \leq 1/6.$$

□

Lemma 3 $\Pr[d' > (1 + \epsilon)d] \leq 1/6]$.

Proof: Suppose $d' > (1 + \epsilon)d$, that is $v < \frac{tN}{(1+\epsilon)d}$. This implies that more than t hash values are less than $\frac{tN}{(1+\epsilon)d} \leq (1 - \epsilon/2)tN/d$. We will show that this happens with small probability.

Let X_i be the indicator random variable for the event $h(b_i) < (1-\epsilon/2)tN/d$ and let $Y = \sum_{i=1}^{d} X_i$. We have $\mathbf{E}[X_i] < (1 - \epsilon/2)t/d + 1/N \leq (1 - \epsilon/4)t/d$ (the $1/N$ is for rounding errors). Hence $\mathbf{E}[Y] \leq (1 - \epsilon/4)t$. As we argued $d' > (1 + \epsilon)d$ happens only if $Y > t$. By Cheybyshev,

$$\Pr[Y > t] \leq \Pr[|Y - \mathbf{E}[Y]| \geq \epsilon t/4] \leq \frac{16\mathbf{Var}[Y]}{\epsilon^r t^2} \leq 16/(\epsilon^2 t^2) \leq 1/6.$$

□

Bibliographic Notes: Our description of the AMS algorithm is taken from notes of Amit Chakrabarti. Flajolet and Martin proposed the basic hash function based algorithm in [5]. [2] Kane, Nelson and Woodruff [7] described an (ϵ, δ_0)-approximation for a fixed δ_0 in space $O(\frac{1}{\epsilon^2} + \log n)$ (the time per update is also the same). This is a theoretically optimum algorithm since there are lower bounds of $\Omega(\epsilon^2)$ and $\Omega(\log n)$ on the space required for an ϵ-approximation. We refer the reader to [4, 6] for analysis and empirical evaluation of an algorithm called HyperLogLog which seems to very well in practice.

References

[1] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58(1):137–147, 1999. Preliminary version in *Proc. of ACM STOC* 1996.

[2] Ziv Bar-Yossef, TS Jayram, Ravi Kumar, D Sivakumar, and Luca Trevisan. Counting distinct elements in a data stream. In *Randomization and Approximation Techniques in Computer Science*, pages 1–10. Springer, 2002.

[3] J Lawrence Carter and Mark N Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143–154, 1979. Preliminary version in *Proc. ACM STOC*, 1977.

[4] Philippe Flajolet, Eric Fusy, Olivier Gandouet, Frédéric Meunier, et al. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. *Analysis of Algorithms 2007 (AofA07)*, pages 127–146, 2007.

[5] Philippe Flajolet and G Nigel Martin. Probabilistic counting algorithms for data base applications. *Journal of computer and system sciences*, 31(2):182–209, 1985.

[6] Stefan Heule, Marc Nunkesser, and Alex Hall. Hyperloglog in practice: Algorithmic engineering of a state of the art cardinality estimation algorithm. In *Proceedings of the EDBT 2013 Conference*, Genoa, Italy, 2013.

[7] Daniel M Kane, Jelani Nelson, and David P Woodruff. An optimal algorithm for the distinct elements problem. In *Proceedings of the twenty-ninth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 41–52. ACM, 2010.