CSE 522: Sublinear (and Streaming) Algorithms **Spring 2014**

Lecture 2: Hashing, Estimating the Number of Distinct Elements

April 2, 2014

Lecturer: Paul Beame *Scribe: Paul Beame*

In the last class we saw that by sample $s = O(\sqrt{n/\epsilon})$ elements and checking whether they contain a duplicate, we can distinguish between whether the input from $[M]^n$ is all distinct or has $< (1-\varepsilon)n$ distinct elements.

We complete the analysis by considering the time and space required to do the computation. One could store the entire set of samples and sort it to check duplicates in time $O(s \log s)$, or, more generally, use some form of dictionary data structure that allows dynamic insertions in order to detect duplicates as soon as they are sampled. This can all be done in $\tilde{O}(\sqrt{n/\varepsilon})$ time and space (where we use $\tilde{O}$ to hide logarithmic factors). In each of these cases one will need at least $\log_2 \binom{M}{s-1}$ bits which is $\Omega(s \log(M/s)$ in order to represent which (distinct) values have already been seen before the last input is read. In particular, this is $\Omega(\sqrt{n/\varepsilon})$.

In the case of data stream algorithms that read the inputs once in order, we will see that we can use much less than $\tilde{\Theta}(\sqrt{n/\varepsilon})$ space to do much more, namely approximate the *number* of distinct elements within a $1 \pm \varepsilon$ multiplicative factor. Observe that such an algorithm will distinguish between n and $< (1-\varepsilon)n$. A key tool for this algorithm and many other data stream algorithms will be to use random hash functions so we review their properties.

1 Random Hash Function Families

We will consider families of functions H such that every $h \in H$ is of the form $h : [M] \to \{0,1\}^\ell$. For convenience we assume that $M = 2^m$ for some integer m and identify $[M]$ with $\{0,1\}^m$. Every such set H induces a uniform distribution on its members and we use $h \sim H$ to denote that h is chosen uniformly and randomly from H.

Definition 1.1. *Hash function family H is* uniform *iff*

$$\mathbb{P}_{h\sim H}[h(j) = v] = \frac{1}{2^\ell} \quad \textit{for all } j \in [M],\ v \in \{0,1\}^\ell.$$

Further, H is 2-universal *iff*

$$\mathbb{P}_{h\sim H}[h(i) = h(j)] = \frac{1}{2^\ell} \quad \textit{for all } i \neq j \in [M].$$

H is pairwise independent *iff H is both uniform and 2-universal.*
More generally, H is k-wise independent *iff for every $A \subseteq [M]$ with $|A| \leq k$ and all $(v_j)_{j \in A}$ with $v_j \in \{0,1\}^\ell$,*

$$\mathbb{P}_{h \sim H}[h(j) = v_j, \textit{ for all } j \in A] = \frac{1}{2^{\ell \cdot |A|}}.$$

(Observe that the property for $|A| = k$ implies the property for all smaller sets.)

The following are some examples of hash function families for k-wise independence.

First consider the case of $h : [M] \to [M]$; i.e., $h : \{0,1\}^m \to \{0,1\}^m$. Let $\mathbb{F}_{2^m}$ be the finite field of 2^m elements[1] Define H to consist of all $h : \mathbb{F}_{2^m} \to \mathbb{F}_{2^m}$ given by degree $k-1$ polynomials:

$$h(x) = \alpha_0 + \alpha_1 x + \cdots + \alpha_{k-1} x^{k-1}$$

for $\alpha_0, \ldots, \alpha_{k-1} \in \mathbb{F}_{2^m}$ where the computation is over $\mathbb{F}_{2^m}$. Observe that it takes $km = k \log_2 M$ bits to represent an element of H.

To see that the family H is pairwise independent, suppose that $A = \{j_1, \ldots, j_k\}$ and let $v_1, \ldots, v_k$ be any elements of $\mathbb{F}_{2^m}$. Observe that the simultaneous equations $h(j_1) = v_1, \ldots, h(j_k) = v_k$ are equivalent to the matrix equation

$$\begin{bmatrix} 1 & j_1 & j_1^2 & \cdots & j_1^{k-1} \\ 1 & j_2 & j_2^2 & \cdots & j_2^{k-1} \\ \cdot & \cdot & \cdot & \cdots & \cdot \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & j_k & j_k^2 & \cdots & j_k^{k-1} \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_{k-1} \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \\ \cdot \\ \vdots \\ v_k \end{bmatrix}.$$

The matrix on the left is called a Vandermonde matrix and has determinant equal to $\prod_{i \neq i'}(j_i - j_{i'})$ which is non-zero since all the j_i are distinct in the field. Therefore using its inverse, for any choice of $v_1, \ldots, v_k$, we can solve for a unique choice of the unknowns $\alpha_0, \ldots, \alpha_{k-1}$ that satisfies the equation. Therefore the probability that the equation is satisfied is precisely $1/2^{mk}$ which is what we require.

Notice that in the case of pairwise independence we have a hash function of the familiar form $h(x) = ax + b$ and so this is a natural generalization to k-wise independence of our usual constructions. Also, observe that if we want a k-wise independent construction for $\ell < m$ then we can simply ignore $m - \ell$ bits of the output.

One very nice simple way that we can construct a 2-universal hash function family from $\{0,1\}^n \to \{0,1\}^\ell$, given by Dietzfelbinger et al., is to identify the input with an integer and use ordinary

[1]The field $\mathbb{F}_p$ for prime p consists of the integers modulo p with ordinary addition and multiplication. The elements of $\mathbb{F}_{2^m}$ are not defined using mod in the same way. They are a vector space of polynomials of degree $m-1$ over $\mathbb{F}_2$ and are described using any irreducible polynomial P of degree m over $\mathbb{F}_2$. Operations in $\mathbb{F}_{2^m}$ are polynomial addition modulo 2 and polynomial multiplication taken modulo P and modulo 2.

integer multiplication: That is, we choose a to be an $(m+\ell)$-bit integer and b to be an m-bit integer. If x is an m-bit integer, then the product $ax+b$ will be $(2m+\ell)$-bit integer. The hash function value consists of the middle ℓ bits of this output.

We will very frequently use 2-universal and pairwise independent hash function families but we will see that larger independence will also sometimes be useful.

Pairwise independence The following proposition, which we will frequently apply together with Chebyshev's inequality, is a key to why pairwise independence is so useful:

Proposition 1.2. *If $X = X_1 + \cdots + X_n$ and the X_i are pairwise independent then*

$$Var(X) = Var(X_1) + \cdots + Var(X_n).$$

Further, if each $X_i \in \{0,1\}$ then

$$Var(X) \leq \mathbb{E}(X).$$

Proof. By definition,

$$\begin{aligned}
Var(X) &= \mathbb{E}(X^2) - \mathbb{E}(X)^2 \\
&= \mathbb{E}((X_1 + \cdots + X_n)^2) - (\mathbb{E}(X_1) + \cdots \mathbb{E}(X_n))^2 \\
&= \mathbb{E}(\sum_{i,j} X_i X_j) - \sum_{i,j} \mathbb{E}(X_i)\mathbb{E}(X_j) \\
&= \sum_{i,j} (\mathbb{E}(X_i X_j) - \mathbb{E}(X_i)\mathbb{E}(X_j)) \\
&= \sum_{i} (\mathbb{E}(X_i^2) - \mathbb{E}(X_i)^2) + \sum_{i \neq j} (\mathbb{E}(X_i X_j) - \mathbb{E}(X_i)\mathbb{E}(X_j)) \\
&= \sum_{i} (\mathbb{E}(X_i^2) - \mathbb{E}(X_i)^2) \qquad \text{by pairwise indepence} \\
&= \sum_{i} Var(X_i).
\end{aligned}$$

Now if each $X_i \in \{0,1\}$, then

$$\sum_i (\mathbb{E}(X_i^2) - \mathbb{E}(X_i)^2) = \sum_i (\mathbb{E}(X_i) - \mathbb{E}(X_i)^2) \leq \sum_i \mathbb{E}(X_i) = E(X).$$

□

Thus, in particular, if X is the sum of pairwise independent indicator random variables, Chebyshev's inequality implies that

$$\mathbb{P}[|X - \mathbb{E}(X)| \geq c\sqrt{\mathbb{E}(X)}] \leq \frac{Var(X)}{c^2 \mathbb{E}(X)} \leq 1/c^2,$$

which is a fairly sharp bound on the deviation of X from its expectation.

2 Estimating the Number of Distinct Elements in a Data Stream

For a data stream $x_1, \ldots, x_n \in [M]$ we will find it useful to define the *frequency vector* for x,

$$f = (f_1, \ldots, f_M) \qquad \text{where } f_j = \#\{i \mid x_i = j\}.$$

The number of distinct elements in the stream is therefore $\#\{j \mid f_j > 0\}$.

Another way to express the number of distinct elements is as

$$F_0 = \sum_{j=1}^{M} f_j^0$$

where we take $0^0 = 0$. We will show how to estimate F_0 for a data stream using roughly logarithmic space. Later we will show a $(1 \pm \epsilon)$ factor approximation, but we begin with a simple algorithm that achieves a constant-factor approximation.

Basic Idea If we apply a random hash function $h : [M] \to [M]$ to the elements of the input then their values will look like roughly F_0 random elements of M. The algorithm can then maintain a simple property of the values seen in order to estimate the value of F_0. The property suggested by [Flajolet-Martin 85] is to maintain the largest number of leading (or trailing) 0's seen in the input since the probability of seeing a pattern of t such 0's is 2^{-t}. We will see that pairwise independence is enough. ([Alon-Matias-Szegedy 96] gave a small variant of this using the smallest hash value seen.)

For $j \in [M]$, define $\mathrm{zeroes(j)} = \max\{\mathrm{r} \mid 2^{\mathrm{r}} \text{ divides j}\}$, which is the number of trailing 0's in the binary expansion of j.

Basic F_0 Estimation Algorithm

Initialize: Choose $h : [M] \to [M]$ from H a pairwise independent family of hash functions
$z \leftarrow 0$
Process:
for each i **do**
    $z \leftarrow \max(\mathrm{zeroes}(h(x_i), z)$
end for
Output:
$2^{z+1/2}$ (geometric mean of 2^z and 2^{z+1})

It takes $2 \log_2 M$ bits to represent h and $\log \log n$ bits for z so the total space is $O(\log M + \log \log n)$. We will see that with constant probability, the output of this algorithm is between $F_0/3$ and $3F_0$ and then see how to improve the success probability to $1 - \delta$ for any $\delta > 0$.