Sublinear Time and Space Algorithms 2016B – Lecture 2
Distinct Elements, Point Queries and Hash Functions*

Robert Krauthgamer

1 Distinct Elements

Problem Definition: Let $x \in \mathbb{R}^n$ be the frequency vector of the input stream, and let $\|x\|_0 = |\{i \in [n] : x_i > 0\}|$ be the number of distinct elements in the stream. It's also called the F_0-moment of σ.

Naive algorithms: Storage $O(n)$ (a bit for each possible item) or $O(m \log n)$ (list of seen items) bits.

Algorithm FM [Flajolet and Martin, 1985]:

It employs a "hash" function $h : [n] \to [0, 1]$ where each $h(i)$ has an independent uniform distribution on $[0, 1]$. (This is an "idealized" description, because even though we can generate n truly random bits, we cannot store and re-use them.)

Idea: We will have exactly $\mathrm{d}^* = \|x\|_0$ distinct hashes, and since they are random, by symmetry their minimum should be at $1/(\mathrm{d}^* + 1)$.

1. Init: $z = 1$
2. When item $i \in [n]$ is seen, update $z = \min\{z, h(i)\}$
3. Output: $1/z - 1$

Storage requirement: $O(1)$ words (not including randomness); we will discuss implementation issues later.

Denote by $\mathrm{d}^* := \|x\|_0$ the true value, and let Z denote the final value of z (to emphasize it is a random variable).

Lemma 1: $\mathbb{E}[Z] = 1/(\mathrm{d}^* + 1)$.

Note: This is the expectation of Z and not of its inverse $1/Z$ (as used in the output).

Proof: Formally, we use a trick to avoid the integral calculation (which is actually straightfor-

*These notes summarize the material covered in class, usually skipping proofs, details, examples and so forth, and possibly adding some remarks, or pointers. The exercises are for self-practice and need not be handed in. In the interest of brevity, most references and credits were omitted.

ward). Choose an additional random value X uniformly from $[0,1]$ (for sake of analysis only), then by the law of total expectation

$$\mathbb{E}[Z] = \mathop{\mathbb{E}}_{Z}[\Pr_{X}[X < Z \mid Z]] = \mathop{\mathbb{E}}_{Z}[\mathop{\mathbb{E}}_{X}[\mathbb{1}_{\{X<Z\}} \mid Z]] = \mathbb{E}[\mathbb{1}_{\{X<Z\}}] = 1/(\mathrm{d}^* + 1).$$

Lemma 2: $\mathbb{E}[Z^2] = \frac{2}{(d+1)(d+2)}$ and thus $\mathrm{Var}[Z] \leq (\mathbb{E}[Z])^2$.

Exer: Prove this lemma using the above trick with two new random values (and/or prove both by calculating the integral).

Algorithm FM+:

1. Run $k = O(1/\varepsilon^2)$ independent copies of algorithm FM, keeping in memory $Z_1, \ldots, Z_k$ (and functions $h^1, \ldots, h^k$)

2. Output: $1/\bar{Z} - 1$ where $\bar{Z} = \frac{1}{k}\sum_{i=1}^{k} Z_i$

As before, averaging reduces the standard deviation by factor $\sqrt{k}$, and then by Chebyshev's inequality, WHP $\bar{Z} \in \mathrm{d}^* \pm O(\mathrm{d}^* / \sqrt{k}) = \mathrm{d}^* \pm \varepsilon\, \mathrm{d}^*$.

Storage requirement: $O(k)$ words (not including randomness); we will discuss implementation issues later.

Remark: The storage can be improved similarly to the probabilistic counting. It suffices to store a $(1+\varepsilon)$-approximation of z, which can reduce the number of bits from $O(\log n)$ (in a "typical" implementation of the real-valued hashes) to $O(\log \log n)$. A particularly efficient 2-approximation is to store the number of zeros in the beginning of z's binary representation.

Remark: Notice this algorithm does not work under deletions.

2 Alternative algorithm for Distinct Elements

Algorithm Bottom k [Bar Yossef, Jayram, Kumar, Sivakumar, and Trevisan, 2002]:

Idea: Use only one hash function, and store the k smallest values seen.

1. Init: $z_1 = \cdots = z_k = 1$

2. When item $i \in [n]$ is seen, update $z_1 < \cdots < z_k$ to be the k smallest distinct values among $\{z_1, \ldots, z_k, h(i)\}$

3. Output: $X := k/z_k$

Storage requirement: Again, $O(k)$ words (not including randomness); we will discuss implementation issues later.

Remark: Notice the output will not make sense if $k > \mathrm{d}^*$, because z_k will maintain its initial value of 1. Figure out where this is needed in the analysis.

Lemma 3: For suitable $k = O(1/\varepsilon^2)$,

$$\Pr[X > (1+\varepsilon)\,\mathrm{d}^*] \le 0.05,$$
$$\Pr[X < (1-\varepsilon)\,\mathrm{d}^*] \le 0.05.$$

Thus, $X \in (1 \pm \varepsilon)\,\mathrm{d}^*$ with probability $\ge 90\%$.

Intuition: The event $X = k/z_k > (1+\varepsilon)\,\mathrm{d}^*$ is equivalent to $z_k < \frac{k}{(1+\varepsilon)\,\mathrm{d}^*}$, which means that at least k hashes are smaller than some threshold, while each of the d^* distinct hashes seen meets this threshold independently with probability $\frac{k}{(1+\varepsilon)\,\mathrm{d}^*}$, hence we expect only $\frac{k}{1+\varepsilon}$ hashes to meet the threshold. If we set $k \ge 1/\varepsilon^2$, then the standard deviation is $\sqrt{k} \le \varepsilon k$, and we can use Chebyshev's inequality.

Exer: Prove the above lemma.

3 ℓ_1 Point Query via CountMin

Problem Definition: Let $x \in \mathbb{R}^n$ be the frequency vector of the input stream, and let $\|x\|_p = (\sum_i |x_i|^p)^{1/p}$ be its ℓ_p-norm. Let $\alpha \in (0,1)$ and $p > 0$ be parameters known in advance.

The goal is to estimate every coordinate with additive error, namely, given query $i \in [n]$, report $\tilde{x}_i$ such that WHP

$$\tilde{x}_i \in x_i \pm \alpha \|x\|_p.$$

Observe: $\|x\|_1 \ge \|x\|_2 \ge \ldots \ge \|x\|_\infty$, hence higher norms (larger p) give better accuracy. We will see an algorithm for ℓ_1, which is the easiest.

Exer: Show that the ℓ_1 and ℓ_2 norms differ by at most a factor of $\sqrt{n}$, and that this is tight. Do the same for ℓ_2 and ℓ_∞.

It is not difficult to see that ℓ_∞ point query is hard. For instance, with $\alpha = 1/2$ we could recover an arbitrary binary vector $x \in \{0,1\}^n$, which (at least intuitively) requires $\Omega(n)$ bits to store.

Theorem 4 [Cormode-Muthukrishnan, 2005]: There is a streaming algorithm for ℓ_1 point queries that uses a (linear) sketch of $O(\alpha^{-1} \log n)$ memory words to achieve accuracy α with success probability $1 - 1/n^2$.

We will initially assume all $x_i \ge 0$.

Algorithm CountMin:

(Assume all $x_i \ge 0$.)

1. Init: Set $w = 4/\alpha$ and choose a random hash function $h : [n] \to [w]$.

2. Update: Maintain table/vector $S = [S_1, \ldots, S_w]$ where $S_j = \sum_{i:h(i)=j} x_i$.

3. Output: To estimate x_i return $\tilde{x}_i = S_{h(i)}$.

The update step can indeed be implemented in a streaming fashion: When item i arrives, we need to update $x \leftarrow x + e_i$. This update is easy because the sketch is a linear map $S : \mathbb{R}^n \to \mathbb{R}^w$ (observe that $S_j = \sum_i \mathbb{1}_{\{h(i)=j\}} x_i$), and thus $S(x + e_i) = S(x) + S(e_i)$.

We call S a sketch to emphasize it is a succinct version of the input.

Analysis (correctness): We saw in class that $\tilde{x}_i \geq x_i$ and $\Pr[\tilde{x}_i \geq x_i + \alpha \|x\|_1] \leq 1/4$.

Algorithm CountMin+:

1. Run $t = \log n$ independent copies of algorithm CountMin, keeping in memory the vectors $S^1, \ldots, S^t$ (and functions $h^1, \ldots, h^t$)

2. Output: the minimum of all estimates $\hat{x}_i = \min_l S^l_{h^l(i)}$

Analysis (correctness): As before, $\hat{x}_i \geq x_i$ and

$$\Pr[\hat{x}_i > x_i + \alpha \|x\|_1] \leq (1/4)^t = 1/n^2.$$

By a union bound, with probability at least $1 - 1/n$, for all $i \in [n]$ we will have $x_i \leq \hat{x}_i \leq x_i + \alpha \|x\|_1$.

Space requirement: $O(\alpha^{-1} \log n)$ words (for success probability $1 - 1/n^2$), without counting memory used to represent/store the hash functions.

General x (allowing negative entries):

Algorithm CountMin actually extends to general x that might be negative, and achieves the guarantee

$$\Pr[\tilde{x}_i \in x_i \pm \alpha \|x\|_1] \leq 1/4.$$

Exer: complete the proof.

But now to amplify the success probability, we use median instead of minimum.

Chernoff-Hoeffding concentration bounds: Let $X = \sum_{i \in [n]} X_i$ where $X_i \in [0, 1]$ for $i \in [n]$ are independently distributed random variables. Then

$$\begin{aligned}
\forall t > 0, \qquad & \Pr[|X - \mathbb{E}[X]| \geq t] \leq 2e^{-2t^2/n}. \\
\forall 0 < \varepsilon \leq 1, \qquad & \Pr[X \leq (1 - \varepsilon)\,\mathbb{E}[X]] \leq e^{-\varepsilon^2\,\mathbb{E}[X]/2}. \\
\forall 0 < \varepsilon \leq 1, \qquad & \Pr[X \geq (1 + \varepsilon)\,\mathbb{E}[X]] \leq e^{-\varepsilon^2\,\mathbb{E}[X]/3}. \\
\forall t \geq 2e\,\mathbb{E}[X], \qquad & \Pr[X \geq t] \leq 2^{-t}.
\end{aligned}$$

Algorithm CountMin++:

1. Run $k = O(\log n)$ independent copies of algorithm CountMin, keeping in memory the vectors $S^1, \ldots, S^k$ (and functions $h^1, \ldots, h^k$)

2. Output: To estimate x_i report the median of all basic estimates $\hat{x}_i = \text{median}\{S^l_{h^l(i)} : l \in [k]\}$

Exer: Prove that

$$\Pr[\hat{x}_i \in x_i \pm \alpha \|x\|_1] \leq 1/n^2.$$

Hint: Define an indicator Y_j for the event that copy $j \in [k]$ succeeds, then use one of the concentration bounds.

Exer: Use these concentration bounds to amplify the success probability of the algorithms we saw for Distinct Elements and for Probabilistic Counting (say from constant to $1 - 1/n^2$).

Hint: use independent repetitions + median.

4 Hash Functions

Recall that two (discrete) random variables X, Y are independent if

$$\forall x, y \qquad \Pr[X = x, Y = y] = \Pr[X = x] \cdot \Pr[Y = y].$$

This is equivalent to saying that the conditioned random variable $X|Y$ has exactly the same distribution as X. In particular, it implies $\mathbb{E}[XY] = \mathbb{E}[X] \cdot \mathbb{E}[Y]$.

Pairwise independent random variables: A collection of random variables $X_1, \ldots, X_n$ is called *pairwise independent* if for all $i \neq j \in [n]$, the variables X_i and X_j are independent.

Example: Let $X, Y \in \{0, 1\}$ be random and independent bits, and let $Z = X \oplus Y$. Then X, Y, Z are clearly not mutually (fully) independent, but they are pairwise independent.

Observation: When $X_1, \ldots, X_n$ are pairwise independent, the variance $\mathrm{Var}(\sum_i X_i)$ is exactly the same as if they were fully independent, because

$$\mathrm{Var}(\sum_i X_i) = \mathbb{E}[(\sum_i X_i)^2] - (\mathbb{E}[\sum_i X_i])^2 = \sum_{i,j} \mathbb{E}[X_i X_j] - (\sum_i \mathbb{E}[X_i])^2.$$

A different way to see it, is via the following well-known (and easy) fact: If $X_1, \ldots, X_n$ are pairwise independent (and have finite variance), then $\mathrm{Var}(\sum_i X_i) = \sum_i \mathrm{Var}(X_i)$.

Pairwise independent hash family: A family H of hash functions $h : [n] \to [M]$ is called *pairwise independent* if for all $i \neq j \in [n]$,

$$\forall x, y \qquad \Pr_{h \in H}[h(i) = x, h(j) = y] = \Pr[h(i) = x] \Pr[h(j) = y].$$

A common scenario is that each $h(i)$ is uniformly distributed over $[M]$.

Universal hashing: A family H of hash functions $h : [n] \to [M]$ is called *2-universal* if for all $i \neq j \in [n]$,

$$\forall x, y \qquad \Pr_{h \in H}[h(i) = x, h(j) = y] \leq 1/M.$$

Observe that 2-universality is a weaker requirement that pairwise independence, but it suffices for many algorithms.

Construction of pairwise independent hashing:

Assume $M \geq n$ and that M is a prime number (if not, we can pick a larger M that is a prime). Pick random $p, q \in \{0, 1, 2, \ldots, M-1\} = [M]$ and set accordingly $h_{p,q}(i) = pi + q \pmod{M}$.

The family $H = \{h_{p,q} : p, q\}$ is pairwise independent because for all $i \neq j$ and all x, y,

$$\Pr_{h \in H}[h(i) \equiv x, h(j) \equiv y] = \Pr_{p,q}\left[\begin{pmatrix} i & 1 \\ j & 1 \end{pmatrix}\begin{pmatrix} p \\ q \end{pmatrix} \equiv \begin{pmatrix} x \\ y \end{pmatrix}\right] = \Pr_{p,q}\left[\begin{pmatrix} p \\ q \end{pmatrix} \equiv \begin{pmatrix} i & 1 \\ j & 1 \end{pmatrix}^{-1}\begin{pmatrix} x \\ y \end{pmatrix}\right] = \frac{1}{M^2},$$

where we relied on the above matrix being invertible.

Storing a function $h_{p,q}$ from this family can be done by storing p, q, which requires $\log |H| = O(\log M)$ bits. In general, $\log |H|$ bits suffice to store an index of $h \in H$.

Exer: Show that the correctness of algorithm CountMin (for ℓ_1 point query) extends to using a universal hash function, and analyze how much additional storage the hash function requires.

Exer: Show that the correctness of algorithm Bottom k (for Distinct Elements) can be extended to using a pairwise independent hash function $h : [n] \to [n^3]$ (instead of continuous range $[0, 1]$), and analyze how much additional storage the hash function requires.

Hint: Our analysis used events of the form $\{h(i) < \mathit{threshold}\}$, and relied on independence for every pair $h(i), h(j)$.