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 $\sigma$.

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 $h : [n] \rightarrow [0,1]$ 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 $d^* = \|x\|_0$ distinct hashes, and since they are random, by symmetry their minimum should be around $1/(d^* + 1)$.

1. Init: $z = 1$ and a hash function $h$
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 $d^* := \|x\|_0$ the true value, and let $Z$ denote the final value of $z$ (to emphasize it is a random variable).

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

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

Proof: We will use a trick to avoid the integral calculation (which is actually straightforward).
Choose an additional random value $X$ uniformly from $[0, 1]$ (for sake of analysis only), then by the law of total expectation

$$E[Z] = E\left[\frac{Z}{X} \Pr[X < Z | Z]\right] = E\left[E\left[\mathbb{1}_{\{X < Z\}} | Z\right]\right] = E\left[\mathbb{1}_{\{X < Z\}}\right] = 1/(d^* + 1).$$

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

**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: $1/\overline{Z} - 1$ where $\overline{Z} = \frac{1}{k} \sum_{i=1}^{k} Z_i$

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

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: $z_1 = \cdots = z_k = 1$ for $k = O(1/\varepsilon^2)$ and a hash function $h$
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 $k > d^*$, because $z_k$ will maintain its initial value of 1. Figure out where this is needed in the analysis.
Lemma 3: For suitable $k = O(1/\varepsilon^2)$,
\[
\Pr[X > (1 + \varepsilon) d^*] \leq 0.05,
\]
\[
\Pr[X < (1 - \varepsilon) d^*] \leq 0.05.
\]
Thus, $X \in (1 \pm \varepsilon) d^*$ with probability $\geq 90\%$.

Intuition: The event $X = k/z_k > (1 + \varepsilon) d^*$ is equivalent to $z_k < \frac{k}{(1+\varepsilon)d^*}$, which means that at least $k$ hashes are smaller than some threshold; since each of the $d^*$ distinct hash values meets this threshold independently with probability $\frac{k}{(1+\varepsilon)d^*}$, we expect only $\frac{k}{1+\varepsilon}$ hashes to meet the threshold.

If we set $k \geq 1/\varepsilon^2$, then the standard deviation is $\sqrt{k} \leq \varepsilon k$, and we can use Chebyshev’s inequality.

Exer: Prove the above lemma.

3 \quad \ell_1 \text{ 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 $\ell_p$-norm. Let $\alpha \in (0, 1)$ and $p \geq 1$ 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: $\|x\|_1 \geq \|x\|_2 \geq \ldots \geq \|x\|_\infty$, hence higher norms (larger $p$) give better accuracy. We will see an algorithm for $\ell_1$, which is the easiest.

Exer: Show that the $\ell_1$ and $\ell_2$ norms differ by at most a factor of $\sqrt{n}$, and that this is tight. Do the same for $\ell_2$ and $\ell_\infty$.

It is not difficult to see that $\ell_\infty$ 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 $\ell_1$ point queries that uses a (linear) sketch of $O(\alpha^{-1} \log n)$ memory words to achieve accuracy $\alpha$ with success probability $1 - 1/n^2$.

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

Algorithm CountMin:

(Assume all $x_i \geq 0$.)

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 $L : \mathbb{R}^n \to \mathbb{R}^w$ (observe
that $S_j = \sum_i 1_{\{h(i) = j\}}x_i)$, and thus $L(x + e_i) = L(x) + L(e_i)$.

We call $S$ a sketch to emphasize it is a succinct version of the input, and $L$ a sketching matrix.

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