1 Distinct Elements

We will continue our topic on streaming algorithms. The first problem we will talk about today is the Distinct Elements problem. The input is a stream of elements \((a_1, \ldots, a_n)\) where each \(a_i \in [U]\). Let \(F\) denote the number of distinct elements in the input, e.g., \((1, 3, 4, 1, 3)\) has three distinct elements \(\{1, 3, 4\}\), and \((1, 1, 1, 4)\) has two \(\{1, 4\}\). The problem asks to process the stream using small space and output an estimate \(\hat{F} = (1 \pm \varepsilon)F\) with probability \(1 - \delta\).

The naive algorithm simply stores all distinct elements it has seen so far. In worst-case, all elements of the stream may be distinct, i.e., the algorithm must use \(O(n)\) space. When there is a space restriction of \(S \ll n\), one other natural thought is to sample a subset of the stream, count how many distinct elements there are, and scale properly. However, it has low accuracy: consider a stream where \((1 - \gamma)\)-fraction is the same element \(x\) (for some very tiny \(\gamma\)), and the rest are all distinct. We are likely to only sample \(x\), which means the algorithm does not even know the existance of the rest of the elements. In particular, it cannot distinguish between such an input, which has very large \(F\), and an all-\(x\) input, which has \(F = 1\). Today, we are going to talk about an algorithm that uses \(O_{\varepsilon, \delta}(\log U)\) bits of space and solves the problem. Note that this space bound for constant \(\varepsilon\) and \(\delta\) is (asymptotically) the same as just storing one element from \([U]\).

The following fact about random variables will guide our algorithm design, although it is not directly used.

**Fact 1.** Let \(X_1, \ldots, X_F\) be independent uniform random variables taking values in \([0, 1]\). Let \(X\) be their minimum, we have
\[
\mathbb{E}[X] = \frac{1}{F + 1}.
\]
Let \(X^{(k)}\) be the \(k\)-th minimum, its expectation is
\[
\mathbb{E}[X^{(k)}] = \frac{k}{F + 1}.
\]

We will assign each distinct element we saw a random number in \([0, 1]\), observe a concrete value of the \(k\)-th minimum, and use the above relation to estimate the number of variables \(F\).

### 1.1 An “ideal” algorithm

Now we describe an ideal algorithm that assuming it has access to random hash functions \(h : [U] \rightarrow [0, 1]\), and we are able to “store” real numbers. Later, we will remove the assumptions.

Consider the following algorithm, which is referred to the KMV algorithm (\(k\)-minimum value).
1. fix a parameter \( k \geq 1 \)
2. set \( S \leftarrow \emptyset \) (maintain the smallest \( k \) numbers we see)
3. for \( i = 1, \ldots, n \)
4. \( S \leftarrow S \cup \{h(a_i)\} \)
5. if \(|S| > k\), remove \( \max(S) \) from \( S \)
6. if \(|S| < k\), return \(|S|\); otherwise, return \( \tilde{F} := k / \max(S) \)

If we see less than \( k \) distinct hashes, we return the exact number of distinct numbers.
Otherwise, we use the above fact to estimate \#distinct (note that following the above formula, we should have returned \( k / \max(S) - 1 \), but it is already sufficiently accurate without the “−1”, and this makes the analysis cleaner).

**Analysis** Suppose we have \( F \) distinct numbers, and their hash values are \( V_1, \ldots, V_F \) respectively. Then \( V_1, \ldots, V_F \) are independent random numbers in \([0, 1]\).

Note that \( \tilde{F} > (1 + \varepsilon)F \), if and only if \( \max(S) < k/(1 + \varepsilon)F \), if and only if there are at least \( k \) numbers in \((V_1, \ldots, V_F)\) that are \(< k/(1 + \varepsilon)F \). Below, we upper bound the probability of this event via Chebyshev’s inequality.

For \( i = 1, \ldots, F \), let \( X_i \) indicate if \( V_i < k/(1 + \varepsilon)F \). We have that \( \Pr[X_i = 1] = k/(1 + \varepsilon)F \). Let \( X = X_1 + \cdots + X_F \). By linearity of expectation, we have the following claim.

**Claim 2.** We have \( \mathbb{E}[X] = k/(1 + \varepsilon) \).

We can also bound its variance.

**Claim 3.** We have \( \operatorname{Var}[X] \leq k \).

**Proof.** Since \( X_i \) are independent, we have

\[
\operatorname{Var}[X] = F \cdot \operatorname{Var}[X_1] = F \cdot (k/(1 + \varepsilon)F - (k/(1 + \varepsilon)F)^2) < k.
\]

By Chebyshev’s inequality, we have

\[
\Pr[X \geq k] \leq \Pr[|X - k/(1 + \varepsilon)| \geq \varepsilon k/(1 + \varepsilon)] \leq k/((\varepsilon k/(1 + \varepsilon))^2) \leq O(1/\varepsilon^2 k).
\]

Since \( X \geq k \) if and only if \( \tilde{F} > (1 + \varepsilon)F \), by setting \( k = C \varepsilon^{-2} \) for a large constant \( C \), we have \( \Pr[\tilde{F} > (1 + \varepsilon)F] < 1/8 \) Similarly, we can also prove the same bound on \( \Pr[\tilde{F} < (1 - \varepsilon)F] \).

Therefore, the algorithm outputs an accurate estimate with constant probability by storing \( O(k) = O(\varepsilon^{-2}) \) real numbers.

### 1.2 Median

Similar to what we covered in the last lecture, by repeating the algorithm \( O(\log(1/\delta)) \) times in parallel, and return the median of the estimates, we will have success probability at least \( 1 - \delta \). The overall space usage is \( O(\varepsilon^{-2} \log(1/\delta)) \) numbers.
1.3 Remove the assumptions

The first assumption that we can store real numbers can be removed by discretization. It is not hard to verify that by taking values on the set \( \{1/M, 2/M, \ldots, (M-1)/M, 1\} \), we will have a rounding error of \( \pm 1/M \). By setting \( M = U \), the above algorithm still succeeds with the same probability. Now we only need the hash functions to take values in \([M]\).

The second assumption is that \( h \) is a random hash function \([U] \to [M]\). Observe that in the proof the only step that uses the independence of the hash values is “\( \text{Var}[X] = F \cdot \text{Var}[X_1] \)” . In fact, this step holds when \( h \) is pairwise independent (see Lecture Note 2). It is known that pairwise independent hash families of size \( \text{poly}(U, M) \) exist. That is, a hash function can be represented using \( O(\log(U+M)) \) bits. Therefore, the total space usage is \( O(\varepsilon^{-2} \log(1/\delta) \log U) \).

2 Frequency moments

Consider a stream \((a_1, \ldots, a_n)\) where \( a_i \in [U] \). For any \( x \in [U] \), let \( f_x \) be the number of occurrences of \( x \) in the input. Then the \( p \)-th frequency moment is

\[
F_p = \sum_x f_x^p.
\]

The two streaming algorithms we saw so far solve the \( p = 0 \) case (distinct elements if we treat \( 0^0 = 0 \)) and \( p = 1 \) case (estimate the length of the stream).

Next, we are going to show that \( F_2 \) can also be estimated using small space. Consider the following algorithm, which is usually referred to as the AMS sketch.

1. assume we have access to a random hash function \( \sigma : [U] \to \{-1, 1\} \)
2. set \( X \leftarrow 0 \)
3. for \( i = 1, \ldots, n \)
4. \( X \leftarrow X + \sigma(a_i) \)
5. return \( X^2 \)

Let us first see why this algorithm reasonably estimates \( F_2 \). Fix the input stream, which determines the frequencies \( f_x \), and the hash function \( \sigma \). Then the value of \( X \) is simply

\[
X = \sum_x \sigma(x) \cdot f_x.
\]

Now the value we return \( X^2 \) is equal to

\[
X^2 = \left( \sum_x \sigma(x) \cdot f_x \right)^2 = \sum_{x_1, x_2} \sigma(x_1)\sigma(x_2)f_{x_1}f_{x_2} = \sum_x \sigma(x)^2f_x^2 + 2 \sum_{x_1 < x_2} \sigma(x_1)\sigma(x_2)f_{x_1}f_{x_2}.
\]
First note that $\sigma(x)^2 = 1$, hence, the first term is equal to $F_2$. On the other hand, the second term has expectation 0, since for $x_1 \neq x_2$, $\sigma(x_1)$ and $\sigma(x_2)$ are independent, and we have
\[
E[\sigma(x_1)\sigma(x_2)f_{x_1}f_{x_2}] = 0.
\]
This is stated as the following claim.

**Claim 4.** $E[X^2] = F_2$.

We can also bound its variance.

**Claim 5.** $\text{Var}[X^2] \leq O(F^2_2)$.

**Proof.** We have $\text{Var}[X^2] = E[X^4] - E[X^2]^2$.

\[
E[X^4] = \sum_{x_1,x_2,x_3,x_4} E[\sigma(x_1)\sigma(x_2)\sigma(x_3)\sigma(x_4)]f_{x_1}f_{x_2}f_{x_3}f_{x_4}.
\]

Note that this expectation is nonzero only when $x_1 = x_2 = x_3 = x_4$ or they form two distinct pairs, in which case it is equal to 1. Thus, it is equal to
\[
\sum_x f_x^4 + 6 \sum_{x_1 < x_2} f_{x_1}^2f_{x_2}^2 \leq 3\left(\sum_x f_x^2\right)^2.
\]

The claim holds.

Note that the above analysis only requires $h$ to be 4-wise independent, which can be stored using $O(\log n)$ bits. Thus, by doing “median-of-means”, we can estimate $F_2$ with a $(1 \pm \varepsilon)$-approximation with probability $1 - \delta$ using $O(\varepsilon^{-2} \log(1/\delta) \log n)$ space.