**CS787: Advanced Algorithms**

**Scribe:** Priyananda Shenoy and Shijin Kong **Lecturer:** Shuchi Chawla
**Topic:** Streaming Algorithms(continued) **Date:** 10/26/2007

We continue talking about streaming algorithms in this lecture, including algorithms on getting number of distinct elements in a stream and computing second moment of element frequencies.

## 18.1 Introduction and Recap

Let us review the fundamental framework of streaming. Suppose we have a stream coming in from time 0 to time $T$. At each time $t \in [T]$, the coming element is $a_t \in [n]$. Frequency for any element $i$ is defined as $m_i = |\{j|a_j = i\}|$. We will see a few problems to solve under this framework. For different problems, we expect to have a complexity of only $O(\log n)$ and $O(\log T)$ on either storage or update time. We should only make a few passes over the stream under these strong constraints.

First problem we will solve in this lecture is getting the number of distinct elements. A simple way to do this is to perform random sampling on all elements, maintain statistics over sampling, and then extraploate real number of distinct elements from statistics. For example, if we pick $k$ of the $n$ elements uniformly at random, and count the number of distinct elements in samples, we can roughly estimate the number of distinct elements over all elements by multiplying the result with $n$. However, in many cases where elements are unevenly distributed (e. g. some elements dominate), random sampling with a small $k$ will cause much lower estimation than actual number of distinct elements. In order to be accurate we need $k = \Omega(n)$

The second problem, computing second moment of element frequencies depends on accurate estimation of $m_i$. Again we can apply random sampling in similar way and estimate frequency of each element. But again we will have the problem of inaccuracy in random sampling. Some elements may not be sampled at all.

In order to better resolve those problems, we introduce another two different algorithms which perform better than random sampling based algorithm. Before that, let's define what an *unbiased estimator* is:

**Definition 18.1.1** *An unbiased estimator for quantity $Q$ is a random variable $X$ such that $\mathbf{E}[X] = Q$.*

## 18.2 Number of Distinct Elements

Let's assign $c$ to represent the number of distinct elements. We are going to prove that we can probabilistically distinguish between $c < k$ and $c \geqslant 2k$ by using a single bit of memory. $k$ is related to a hash functions family $H$ where,

$\forall h \in H$, $h : [n] \to [k]$

**Algorithm 1**:

Suppose the bit we have is $b$, initially set $b = 0$.

for some $t \in [T]$, if $h(a_t) = 0$, then set $b = 1$.

Let us compute some event probabilities:

$$\mathbf{Pr}[b = 0] = \left(1 - \frac{1}{k}\right)^c$$

$$\mathbf{Pr}[b = 0 | c < k] = \left(1 - \frac{1}{k}\right)^c \geqslant \left(1 - \frac{1}{k}\right)^k \geqslant \frac{1}{4}$$

$$\mathbf{Pr}[b = 0 | c \geqslant 2k] = \left(1 - \frac{1}{k}\right)^c \leqslant \left(1 - \frac{1}{k}\right)^{2k} \leqslant \frac{1}{e^2} \simeq \frac{1}{7.4}$$

Now we have separation between the cases $c < k$ and $c \geqslant 2k$. In the next step, we can use multiple bits to boost this separation.

**Algorithm 2**:

Maintain $x$ bits $b_0, b_1, \ldots, b_x$ and run $Algorithm1$ independently over each bit.

Next we pick a value between $\frac{1}{4}$ and $\frac{1}{e^2}$, say $\frac{1}{6}$. If $|\{j | b_j = 0\}| > \frac{x}{6}$, output $c < k$, else output $c \geqslant 2k$.

**Claim 18.2.1** *The error probability of Algorithm 2 is $\delta$ if $x = O(\log \frac{1}{\delta})$.*

**Proof:** Suppose $c < k$, then the expected number of bits that are 0 in all $x$ bits is atleast $\frac{x}{4}$.

By Chernoff's bound:

$$\mathbf{Pr}\left[\text{actual number of bits that are zero} < \frac{x}{6}\right] =$$
$$\mathbf{Pr}\left[\text{actual number of bits that are } 0 < \left(1 - \frac{1}{3}\right)\frac{x}{4}\right] \leqslant e^{-\frac{1}{2}\frac{1}{3^2}\frac{x}{4}}$$

Using $x = O(\log \frac{1}{\delta})$ gives us the answser with probability $1 - \delta$. Similarly if $c \geqslant 2k$, we can show a similar bound on $\mathbf{Pr}\left[\text{number of actual bits that are } 0 \geqslant \frac{x}{6}\right]$. ∎

We repeat $\log n$ times, and set $\delta = \frac{\acute{\delta}}{\log n}$ for each time. Then, by union bound, the probability that any run fails is $\leq \log n . \frac{\acute{\delta}}{\log n} = \acute{\delta}$.

To get a $(1 + \epsilon)$ approximation, we would need $O\left(\frac{\log n}{\epsilon^2}\left(\log \log n + \log \frac{1}{\acute{\delta}}\right)\right)$ bits

## 18.3 Computing Second Moment

Recall that the $k^{th}$ moment of the stream is defined as $\mu_k = \sum_{i\in[n]} m_i^k$. We will now discuss an algorithm to find $\mu_2$. This measure is used in many applications as an estimate of how much the frequency varies.

**Algorithm 3**:
**Step 1:** Pick a random variable $Y_i \in_{u.a.r} \{-1, 1\} \forall i \in [n]$
**Step 2:** Let the random variable $Z$ be defined as $Z = \sum_t Y_{a(t)}$
**Step 3:** Define the random variable $X$ as $X = Z^2$
**Step 4:** output $X$

**Claim 18.3.1** *$X$ is an unbiased estimator for $\mu_2$; i.e. the expected value of $X$ equals $\mu_2$*

**Proof:**

Z can be redefined as:

$$Z = \sum_{i\in[n]} m_i Y_i$$
$$X = Z^2 = \sum_i m_i^2 Y_i^2 + 2\sum_{i\neq j} m_i m_j Y_i Y_j$$
$$\mathbf{E}[X] = \sum_i m_i^2 \mathbf{E}\left[Y_i^2\right] + 2\sum_{i\neq j} m_i m_j \mathbf{E}[Y_i Y_j]$$

Since $Y_i \in \{-1, 1\}, Y_i^2 = 1$. Also the second term evaluates to 0 since $Y_i Y_j$ will evaluate to -1 or +1 with equal probability

$$\mathbf{E}[X] = \sum_i m_i^2 = \mu_2 \tag{18.3.1}$$

■

To get an accurate value, the above algorithm needs to be repeated. The following algorithm specifies how Algorithm 3 can be repeated to obtain accurate results.

**Algorithm 4**:
**Step 1:**
FOR $m = 1$ to $k$
Execute Algorithm 3. Let $X_i$ be the output
ENDFOR
**Step 2:** Calculate the mean of $X_i$ $\overline{X} = \frac{(X_1 + X_2 ... + X_k)}{k}$
**Step 3:** output $\overline{X}$

The expected value of $\overline{X}$ is taken as the value of $\mu_2$. We will see what the value of $k$ needs to be to get an accurate answer with high probability. To do this, we will apply Chebychev's bound. Consider the expected value of $X^2$ :

$$\begin{aligned}\mathbf{E}\big[X^2\big] = \mathbf{E}[\sum_i m_i^4 Y_i^4 + 4\sum_{i\neq j} m_i^3 m_j Y_i^3 Y_j + 6\sum_{i\neq j} m_i^2 m_j^2 Y_i^2 Y_j^2 + \\ 12\sum_{i\neq j\neq \acute{i}} m_i^2 m_j m_{\acute{i}} Y_i^2 Y_j Y_{\acute{i}} + 24\sum_{i\neq j\neq \acute{i}\neq \acute{j}} m_i m_j m_{\acute{i}} m_{\acute{j}} Y_i Y_j Y_{\acute{i}} Y_{\acute{j}}]\end{aligned}$$

Assuming that the variables $Y_i$ are 4-way indepedent, we can simplify this to

$$\mathbf{E}\big[X^2\big] = \sum_i m_i^4 + 6\sum_{i\neq j} m_i^2 m_j^2$$

The variance of $X_i$ (as defined in Algorithm 3) is given by

$$\begin{aligned} var[X] &= \mathbf{E}\big[X^2\big] - (\mathbf{E}[X])^2 \\ &= (\sum_i m_i^4 + 6\sum_{i\neq j} m_i^2 m_j^2) - (\sum_i m_i^4 + 2\sum_{i\neq j} m_i^2 m_j^2) \\ &= 4\sum_{i\neq j} m_i^2 m_j^2 \\ &\leq 2(\sum_i m_i^2)^2 \\ &\leq 2\mu_2^2 \\ var[\overline{X}] &= \frac{var[X]}{k} \leq \frac{2\mu_2^2}{k} \end{aligned}$$

By Chebychev's inequality:

$$\begin{aligned}\mathbf{Pr}\big[|\overline{X} - \mu_2| \geq \epsilon\mu_2\big] &\leq \frac{var[\overline{X}]}{\epsilon^2\mu_2^2} \\ &\leq \frac{2\mu_2^2}{k\epsilon^2\mu_2^2} \\ &\leq \frac{2}{k\epsilon^2}\end{aligned}$$

Hence, to compute $\mu_2$ within a factor of $(1\pm\epsilon)$ with probability $(1-\delta)$, we need to run the algorithm $\frac{2}{\delta\epsilon^2}$ times.

### 18.3.1 Space requirements

Lets analyze the space requirements for the given algorithm. In each run on the algorithm, we need $O(\log T)$ space to maintain $Z$. If we explicitly store $Y_i$, we would need $O(n)$ bits, which is too expensive. We can improve upon this by contructing a hash function to generate values for $Y_i$ on the fly. For the above analysis to hold, the hash function should ensure that any group of upto four $Y_i$s are independent( i.e. the hash function belongs to a 4-Way Independent Hash Family). We skip the details of how to construct such a hash family, but this can be done using only $O(\log n)$ bits per hash function.

### 18.3.2 Improving the accuracy: Median of means method

In Algorithm 4, we use the mean of many trials to compute the required value. This has the disadvantage that some inaccurate trials could adversely affect the solution. So we need a large number of samples linear in $\frac{1}{\delta}$ to get reasonable accuracy. Instead of using the mean, the following procedure can be used to get better results. The idea is to take the median of the means of subsamples. The reason this works better is because the median is less sensitive to the outliers in a sample, as compared to the mean.

Group adjacent $X_i$s into $k$ groups of size $\frac{8}{\epsilon^2}$ each. For each group calculate the mean. Then the expected value of $\overline{X}$ is obtained by taking the median of the $k$ means. The total number of samples of $X$ we use are $k\frac{8}{\epsilon^2}$

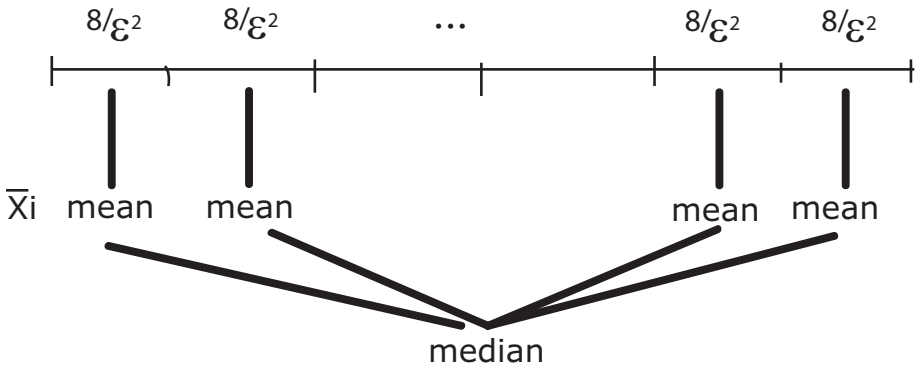


Fig 1: Median of mean method

To see how this improves the accuracy, consider a particular group as shown in the figure. Let $\overline{X_i}$ be the mean of the group.

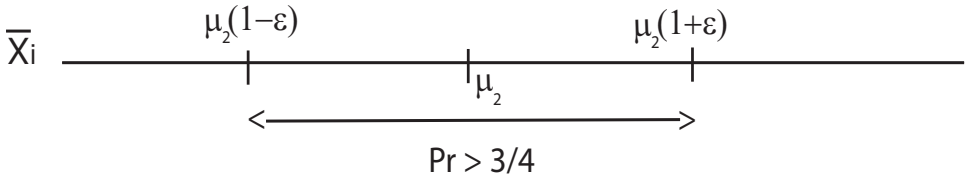


Fig 2: Probability of median falling inside the range

Using Chebychev's Inequality,

$$\begin{aligned}\mathbf{Pr}\left[|\overline{X_i} - \mu_2| > \epsilon\mu_2\right] &\leq \frac{var[X]}{\epsilon^2\mu_2^2} \\ &\leq \frac{2\mu_2^2}{\frac{8}{\epsilon^2}\epsilon^2\mu_2^2} \\ &= \frac{1}{4}\end{aligned}$$

Which essentially means that the probability of a value being outside the interval $[(1-\epsilon)\mu_2, (1+\epsilon)\mu_2]$ is atmost $\frac{1}{4}$. Using Chernoff's bound,

$$\begin{aligned}\mathbf{Pr}[\text{median is outside the interval}] &= \mathbf{Pr}[\text{more than half the samples are outside the interval}] \\ &= e^{-\frac{1}{2}\frac{1}{4^2}k}\end{aligned}$$

If the required probability is $\delta$, we need to pick $k$ so that this value is at most $\delta$, i.e. $k = O(\log \frac{1}{\delta})$ So the number of trials required is $k.\frac{8}{\epsilon^2} = O(\frac{1}{\epsilon^2}log\frac{1}{\delta})$. And the total number of bits used is $O(\frac{1}{\epsilon^2}\log\frac{1}{\delta}(\log T + \log n))$.