Algorithms for Big Data (IV)

Chihao Zhang

Shanghai Jiao Tong University

Oct. 11, 2019

Review of the Last Lecture

Review of the Last Lecture

Last time, we introduced AMS algorithm for counting distinct elements in the streaming model.

Review of the Last Lecture

Last time, we introduced AMS algorithm for counting distinct elements in the streaming model.

We are given a sequence of numbers $\langle a_1, \dots, a_m \rangle$ where each $a_i \in [n]$.

Review of the Last Lecture

Last time, we introduced AMS algorithm for counting distinct elements in the streaming model.

We are given a sequence of numbers $\langle a_1, \ldots, a_m \rangle$ where each $a_i \in [n]$.

It defines a frequency vector $\mathbf{f} = (f_1, \ldots, f_n)$ where $f_i = \left|\left\{k \in [m] \ : \ a_k = i\right\}\right|$.

Review of the Last Lecture

Last time, we introduced AMS algorithm for counting distinct elements in the streaming model.

We are given a sequence of numbers $\langle a_1, \ldots, a_m \rangle$ where each $a_i \in [n]$.

It defines a frequency vector $\mathbf{f} = (f_1, \ldots, f_n)$ where $f_i = \left|\{k \in [m] : a_k = i\}\right|$.

We want to compute the number $d = \left|\{i \in [n] : f_i > 0\}\right|$.

Review of the Last Lecture

Last time, we introduced AMS algorithm for counting distinct elements in the streaming model.

We are given a sequence of numbers $\langle a_1, \dots, a_m \rangle$ where each $a_i \in [n]$.

It defines a frequency vector $\mathbf{f} = (f_1, \dots, f_n)$ where $f_i = \left|\left\{k \in [m] \,:\, a_k = i\right\}\right|$.

We want to compute the number $d = \left|\left\{i \in [n] \,:\, f_i > 0\right\}\right|$.

Algorithm AMS Algorithm for Counting Distinct Elements

Init:

A random Hash function $h : [n] \to [n]$ from a 2-universal family

$Z \leftarrow 0$

On Input y**:**

if $zero(h(y)) > Z$ **then**

  $Z \leftarrow zero(h(y))$

end if

Output:

$\widehat{d} = 2^{Z+\frac{1}{2}}$.

Using $O(\log \frac{1}{\delta} \log n)$ bits of memory, we can obtain

$$\mathbf{Pr}\left[\frac{d}{3} \le \widehat{d} \le 3d\right] \ge 1 - \delta.$$

Using $O(\log \frac{1}{\delta} \log n)$ bits of memory, we can obtain

$$\mathbf{Pr}\left[\frac{d}{3} \le \widehat{d} \le 3d\right] \ge 1 - \delta.$$

We also introduced the BJKST algorithm, a refinement of the AMS algorithm.

Using $O(\log \frac{1}{\delta} \log n)$ bits of memory, we can obtain

$$\mathbf{Pr}\left[\frac{d}{3} \leq \widehat{d} \leq 3d\right] \geq 1 - \delta.$$

We also introduced the BJKST algorithm, a refinement of the AMS algorithm.

We will show today that the BJKST algorithm can produce $\widehat{d}$ which is a $1 \pm \varepsilon$ approximation of d for any $\varepsilon > 0$.

The BJKST Algorithm

The following refinement is due to Bar-Yossef, Jayram, Kumar, Sivakumar and Trevisan.

Algorithm BJKST Algorithm for Counting Distinct Elements

Init: Random Hash functions $h : [n] \to [n]$, $g : [n] \to [b\varepsilon^{-4} \log^2 n]$, both from 2-universal families; $Z \leftarrow 0$, $B \leftarrow \varnothing$

On Input y:

if $zero(h(y)) \geq Z$ **then**

  $B \leftarrow B \cup \{(g(y), zeros(h(y)))\}$

  while $|B| \geq c/\varepsilon^2$ **do**

    $Z \leftarrow Z + 1$

    Remove all (α, β) with $\beta < Z$ from B

  end while

end if

Output: $\widehat{d} = |B|\, 2^Z$

The algorithm maintains a bucket B, which stores those y whose $\mathrm{zeros}(h(y))$ is larger than the current Z.

The algorithm maintains a bucket B, which stores those y whose $\mathrm{zeros}(h(y))$ is larger than the current Z.

We set a cap $L = \frac{c}{\varepsilon^2}$ for the size of B:

The algorithm maintains a bucket B, which stores those y whose $\mathrm{zeros}(h(y))$ is larger than the current Z.

We set a cap $L = \frac{c}{\varepsilon^2}$ for the size of B:

- if $L = \infty$, B stores all entries, and the algorithm is exact;
- if $L = 2$, the algorithm is equivalent to AMS.

The algorithm maintains a bucket B, which stores those y whose $\text{zeros}(h(y))$ is larger than the current Z.

We set a cap $L = \frac{c}{\varepsilon^2}$ for the size of B:

- if $L = \infty$, B stores all entries, and the algorithm is exact;
- if $L = 2$, the algorithm is equivalent to AMS.

Therefore, the size of B is a trade-off between the memory consumption and the accuracy of the algorithm.

Analysis

Analysis

To analyze the algorithm, we first assume that g is simply the identity function from $[n]$ to $[n]$, namely $g(y) = y$ for all $y \in [n]$.

Analysis

To analyze the algorithm, we first assume that g is simply the identity function from $[n]$ to $[n]$, namely $g(y) = y$ for all $y \in [n]$.

We need to store the whole B, whose size is $O\left(\varepsilon^{-2}\right)$.

Analysis

To analyze the algorithm, we first assume that g is simply the identity function from $[n]$ to $[n]$, namely $g(y) = y$ for all $y \in [n]$.

We need to store the whole B, whose size is $O\left(\varepsilon^{-2}\right)$.

Similar to AMS, for every $k \in [n]$, $X_{k,r}$ is the indicator that $h(k)$ has at least r trailing zeros.

Analysis

To analyze the algorithm, we first assume that g is simply the identity function from $[n]$ to $[n]$, namely $g(y) = y$ for all $y \in [n]$.

We need to store the whole B, whose size is $O\left(\varepsilon^{-2}\right)$.

Similar to AMS, for every $k \in [n]$, $X_{k,r}$ is the indicator that $h(k)$ has at least r trailing zeros.

Define $Y_r = \sum_{k \in [n] : f_k > 0} X_{k,r}$ as the number of $h(a_i)$ with trailing zero at least r.

Analysis

To analyze the algorithm, we first assume that g is simply the identity function from $[n]$ to $[n]$, namely $g(y) = y$ for all $y \in [n]$.

We need to store the whole B, whose size is $O\left(\varepsilon^{-2}\right)$.

Similar to AMS, for every $k \in [n]$, $X_{k,r}$ is the indicator that $h(k)$ has at least r trailing zeros.

Define $Y_r = \sum_{k\in[n]:f_k>0} X_{k,r}$ as the number of $h(a_i)$ with trailing zero at least r.

We already know from the last lecture that $\mathbf{E}\left[Y_r\right] = \frac{d}{2^r}$ and $\mathbf{Var}\left[Y_r\right] \le \frac{d}{2^r}$.

If $Z = t$ at the end of the algorithm, then $Y_t = |B|$ and $\widehat{d} = Y_t 2^t$.

If $Z = t$ at the end of the algorithm, then $Y_t = |B|$ and $\widehat{d} = Y_t 2^t$.

We use A to denote the bad event that $\left|Y_t 2^t - d\right| \geq \varepsilon d$, or equivalently

$$\left|Y_t - \frac{d}{2^t}\right| \geq \frac{\varepsilon d}{2^t}.$$

If $Z = t$ at the end of the algorithm, then $Y_t = |B|$ and $\widehat{d} = Y_t 2^t$.

We use A to denote the bad event that $\left|Y_t 2^t - d\right| \geq \varepsilon d$, or equivalently

$$\left|Y_t - \frac{d}{2^t}\right| \geq \frac{\varepsilon d}{2^t}.$$

We will bound the probability of A using the following argument

- if t is small, then $\mathbf{E}[Y_t] = \frac{d}{2^t}$ is large, so we can apply concentration inequalities;

If $Z = t$ at the end of the algorithm, then $Y_t = |B|$ and $\widehat{d} = Y_t 2^t$.

We use A to denote the bad event that $\left|Y_t 2^t - d\right| \geq \varepsilon d$, or equivalently

$$\left|Y_t - \frac{d}{2^t}\right| \geq \frac{\varepsilon d}{2^t}.$$

We will bound the probability of A using the following argument

- if t is small, then $\mathbf{E}\left[Y_t\right] = \frac{d}{2^t}$ is large, so we can apply concentration inequalities;
- the value t is unlikely to be very large.

If $Z = t$ at the end of the algorithm, then $Y_t = |B|$ and $\widehat{d} = Y_t 2^t$.

We use A to denote the bad event that $\left|Y_t 2^t - d\right| \geq \varepsilon d$, or equivalently

$$\left|Y_t - \frac{d}{2^t}\right| \geq \frac{\varepsilon d}{2^t}.$$

We will bound the probability of A using the following argument

- if t is small, then $\mathbf{E}\left[Y_t\right] = \frac{d}{2^t}$ is large, so we can apply concentration inequalities;
- the value t is unlikely to be very large.

We let s be the threshold for small/large value mentioned above.

$$\begin{aligned}
\mathbf{Pr}\left[A\right] &= \sum_{r=1}^{\log n} \mathbf{Pr}\left[\left|Y_r - \frac{d}{2^r}\right| \geq \frac{\varepsilon d}{2^r} \wedge t = r\right] \\
&\leq \sum_{r=1}^{s-1} \mathbf{Pr}\left[\left|Y_r - \frac{d}{2^r}\right| \geq \frac{\varepsilon d}{2^r}\right] + \sum_{r=s}^{\log n} \mathbf{Pr}\left[t = r\right] \\
&= \sum_{r=1}^{s-1} \mathbf{Pr}\left[\left|Y_r - \mathbf{E}\left[Y_r\right]\right| \geq \varepsilon d/2^r\right] + \mathbf{Pr}\left[Y_{s-1} \geq c/\varepsilon^2\right] \\
&\leq \sum_{r=1}^{s-1} \frac{2^r}{\varepsilon^2 d} + \frac{\varepsilon^2 d}{c 2^{s-1}} \leq \frac{2^s}{\varepsilon^2 d} + \frac{\varepsilon^2 d}{c 2^{s-1}}.
\end{aligned}$$

So if we choose s such that $\frac{d}{2^s} = \Theta\left(\varepsilon^{-2}\right)$, $\mathbf{Pr}\left[A\right]$ can be bounded by any constant (depending on c).

$$
\begin{aligned}
\mathbf{Pr}[A] &= \sum_{r=1}^{\log n} \mathbf{Pr}\left[\left|Y_r - \frac{d}{2^r}\right| \geq \frac{\varepsilon d}{2^r} \wedge t = r\right] \\
&\leq \sum_{r=1}^{s-1} \mathbf{Pr}\left[\left|Y_r - \frac{d}{2^r}\right| \geq \frac{\varepsilon d}{2^r}\right] + \sum_{r=s}^{\log n} \mathbf{Pr}[t = r] \\
&= \sum_{r=1}^{s-1} \mathbf{Pr}[|Y_r - \mathbf{E}[Y_r]| \geq \varepsilon d/2^r] + \mathbf{Pr}\left[Y_{s-1} \geq c/\varepsilon^2\right] \\
&\leq \sum_{r=1}^{s-1} \frac{2^r}{\varepsilon^2 d} + \frac{\varepsilon^2 d}{c 2^{s-1}} \leq \frac{2^s}{\varepsilon^2 d} + \frac{\varepsilon^2 d}{c 2^{s-1}}.
\end{aligned}
$$

So if we choose s such that $\frac{d}{2^s} = \Theta\left(\varepsilon^{-2}\right)$, $\mathbf{Pr}[A]$ can be bounded by any constant (depending on c).

Space Complexity

Space Complexity

We need to store

- the function h: $O(\log n)$;
- the function g: $O(\log n)$;
- the bucket B: $O\left(\frac{c}{\varepsilon^2} \cdot \log \mathrm{ran}(g)\right) = O\left(\frac{c}{\varepsilon^2} \log n\right)$.

Space Complexity

We need to store

- the function h: $O(\log n)$;
- the function g: $O(\log n)$;
- the bucket B: $O\left(\frac{c}{\varepsilon^2} \cdot \log \mathrm{ran}(g)\right) = O\left(\frac{c}{\varepsilon^2} \log n\right)$.

The bottleneck is to store B.

Space Complexity

We need to store

- the function h: $O(\log n)$;
- the function g: $O(\log n)$;
- the bucket B: $O\left(\frac{c}{\varepsilon^2} \cdot \log \operatorname{ran}(g)\right) = O\left(\frac{c}{\varepsilon^2} \log n\right)$.

The bottleneck is to store B.

Instead of using identity function g, we can tolerate collisions (with at most constant probability).

Space Complexity

We need to store

- the function h: $O(\log n)$;
- the function g: $O(\log n)$;
- the bucket B: $O\left(\frac{c}{\varepsilon^2} \cdot \log \mathrm{ran}(g)\right) = O\left(\frac{c}{\varepsilon^2} \log n\right)$.

The bottleneck is to store B.

Instead of using identity function g, we can tolerate collisions (with at most constant probability).

This helps to reduce the memory needed (Exercise).

Frequency Estimation

Frequency Estimation

Consider a stream of numbers $\langle a_1, \ldots, a_m \rangle$ and its frequency vector $\mathbf{f} = (f_1, \ldots, f_n)$.

Frequency Estimation

Consider a stream of numbers $\langle a_1, \ldots, a_m \rangle$ and its frequency vector $\mathbf{f} = (f_1, \ldots, f_n)$.

Another fundamental problem is to estimate f_a for each query $a \in [n]$.

Frequency Estimation

Consider a stream of numbers $\langle a_1, \ldots, a_m \rangle$ and its frequency vector $\mathbf{f} = (f_1, \ldots, f_n)$.

Another fundamental problem is to estimate f_a for each query $a \in [n]$.

It is closely related to the `Frequency` problem which asks for the set $\{j : f_j > m/k\}$.

Frequency Estimation

Consider a stream of numbers $\langle a_1, \ldots, a_m \rangle$ and its frequency vector $\mathbf{f} = (f_1, \ldots, f_n)$.

Another fundamental problem is to estimate f_a for each query $a \in [n]$.

It is closely related to the `Frequency` problem which asks for the set $\{j : f_j > m/k\}$.

We now describe a deterministic algorithm for `Frequency-Estimation`.

Misra-Gries

Misra-Gries

Algorithm Misra-Gries Algorithm for Frequency-Estimation

Init: A table A

On Input y:

if $y \in keys(A)$ **then** $A[y] \leftarrow A[y] + 1$

else if $\left|keys(A)\right| \leq k - 1$ **then** $A[j] \leftarrow 1$

else

  for all $\ell \in keys(A)$ **do**

    $A[\ell] \leftarrow A[\ell] - 1$

    if $A[\ell] = 0$ **then**

      Remove ℓ from A

    end if

  end for

end if

Algorithm Misra-Gries (cont'd))

Output: On query j,

if $j \in keys(A)$ **then**

$\widehat{f_j} = A[j]$

else

$\widehat{f_j} = 0$

end if

Analysis

Analysis

The algorithm uses $O(k(\log m + \log n))$ bits of memory.

Analysis

The algorithm uses $O(k(\log m + \log n))$ bits of memory.

It is not hard to see that for each $j \in [n]$, the output $\widehat{f_j}$ satisfies

$$f_j - \frac{m}{k} \le \widehat{f_j} \le f_j.$$

Analysis

The algorithm uses $O\left(k\left(\log m + \log n\right)\right)$ bits of memory.

It is not hard to see that for each $j \in [n]$, the output $\widehat{f_j}$ satisfies

$$f_j - \frac{m}{k} \leq \widehat{f_j} \leq f_j.$$

If $f_j > m/k$, then j is in the table A.

Analysis

The algorithm uses $O(k(\log m + \log n))$ bits of memory.

It is not hard to see that for each $j \in [n]$, the output $\widehat{f_j}$ satisfies

$$f_j - \frac{m}{k} \leq \widehat{f_j} \leq f_j.$$

If $f_j > m/k$, then j is in the table A. The reverse is not correct!

In Misra-Gries, we compute a table A

In *Misra-Gries*, we compute a table A

The table A stores information about the stream, so we can extract frequency from it.

In Misra-Gries, we compute a table A

The table A stores information about the stream, so we can extract frequency from it.

However, Misra-Gries suffers from the following main drawbacks:

In Misra-Gries, we compute a table A

The table A stores information about the stream, so we can extract frequency from it.

However, Misra-Gries suffers from the following main drawbacks:

- given two tables A_1 and A_2 with respect to σ_1 and σ_2 respectively, we don't know how to obtain the table for $\sigma_1 \circ \sigma_2$ (algorithms with this property are called sketches);

In Misra-Gries, we compute a table A

The table A stores information about the stream, so we can extract frequency from it.

However, Misra-Gries suffers from the following main drawbacks:

- given two tables A_1 and A_2 with respect to σ_1 and σ_2 respectively, we don't know how to obtain the table for $\sigma_1 \circ \sigma_2$ (algorithms with this property are called sketches);
- it does not extend to the turnstile model.

In Misra-Gries, we compute a table A

The table A stores information about the stream, so we can extract frequency from it.

However, Misra-Gries suffers from the following main drawbacks:

- given two tables A_1 and A_2 with respect to σ_1 and σ_2 respectively, we don't know how to obtain the table for $\sigma_1 \circ \sigma_2$ (algorithms with this property are called sketches);
- it does not extend to the turnstile model.

In the turnstile model, each entry of the stream is a pair (a_j, Δ_j).

In Misra-Gries, we compute a table A

The table A stores information about the stream, so we can extract frequency from it.

However, Misra-Gries suffers from the following main drawbacks:

- given two tables A_1 and A_2 with respect to σ_1 and σ_2 respectively, we don't know how to obtain the table for $\sigma_1 \circ \sigma_2$ (algorithms with this property are called sketches);
- it does not extend to the turnstile model.

In the turnstile model, each entry of the stream is a pair (a_j, Δ_j).

Upon receiving (a_j, Δ_j), we update f_{a_j} to $f_{a_j} + \Delta_j$.

Count Sketch

Count Sketch

Algorithm Count Sketch

Init:

An array $C[j]$ for $j \in [k]$ where $k = \frac{3}{\varepsilon^2}$.

A random Hash function $h : [n] \to [k]$ from a 2-universal family.

A random Hash function $g : [n] \to \{-1, 1\}$ from a 2-universal family.

On Input (y, Δ):

$C[h(y)] \leftarrow C[h(y)] + \Delta \cdot g(y)$

Output: On query a:

Output $\widehat{f_a} = g(a) \cdot C[h(a)]$.

Analysis

Analysis

Let $X = \widehat{f}_a$ be the output on the query a.

Analysis

Let $X = \widehat{f_a}$ be the output on the query a.

For every $j \in [n]$, let Y_j be the indicator of $h(j) = h(a)$.

Analysis

Let $X = \widehat{f}_a$ be the output on the query a.

For every $j \in [n]$, let Y_j be the indicator of $h(j) = h(a)$.

$$X = g(a) \cdot \sum_{j=1}^{n} f_j \cdot g(j) \cdot Y_j.$$

Analysis

Let $X = \widehat{f_a}$ be the output on the query a.

For every $j \in [n]$, let Y_j be the indicator of $h(j) = h(a)$.

$$X = g(a) \cdot \sum_{j=1}^{n} f_j \cdot g(j) \cdot Y_j.$$

We have

$$\mathbf{E}[X] = \mathbf{E}\left[g(a) \cdot g(a) \cdot f_a \cdot Y_a + \sum_{j \in [n] \setminus \{a\}} g(a) \cdot f_j \cdot g(j) \cdot Y_j \right] = f_a.$$

Analysis

Let $X = \widehat{f}_a$ be the output on the query a.

For every $j \in [n]$, let Y_j be the indicator of $h(j) = h(a)$.

$$X = g(a) \cdot \sum_{j=1}^{n} f_j \cdot g(j) \cdot Y_j.$$

We have

$$\mathbf{E}[X] = \mathbf{E}\left[g(a) \cdot g(a) \cdot f_a \cdot Y_a + \sum_{j \in [n] \setminus \{a\}} g(a) \cdot f_j \cdot g(j) \cdot Y_j\right] = f_a.$$

Let $Z \triangleq \sum_{j \in [n] \setminus \{a\}} f_j \cdot g(a) \cdot g(j) \cdot Y_j$, then $X = f_a + Z$ and $\mathbf{Var}[X] = \mathbf{Var}[Z]$.

Analysis

Let $X = \widehat{f}_a$ be the output on the query a.

For every $j \in [n]$, let Y_j be the indicator of $h(j) = h(a)$.

$$X = g(a) \cdot \sum_{j=1}^{n} f_j \cdot g(j) \cdot Y_j.$$

We have

$$\mathbf{E}[X] = \mathbf{E}\left[g(a) \cdot g(a) \cdot f_a \cdot Y_a + \sum_{j \in [n] \setminus \{a\}} g(a) \cdot f_j \cdot g(j) \cdot Y_j \right] = f_a.$$

Let $Z \triangleq \sum_{j \in [n] \setminus \{a\}} f_j \cdot g(a) \cdot g(j) \cdot Y_j$, then $X = f_a + Z$ and $\mathbf{Var}[X] = \mathbf{Var}[Z]$.

$$
\begin{aligned}
\mathbf{E}\left[Z^2\right] &= \mathbf{E}\left[\sum_{j\in[n]\setminus\{a\}} f_j \cdot g(a) \cdot g(j)\, Y_j\right] \\
&= \mathbf{E}\left[\sum_{j\in[n]\setminus\{a\}} f_j^2 \cdot Y_j^2 + \sum_{j,j'\in[n]\setminus\{a\}:j\neq j;} f_j \cdot f_{j'} \cdot g(j) \cdot g(j') \cdot Y_j \cdot Y_{j'}\right] \\
&= \mathbf{E}\left[\sum_{j\in[n]\setminus\{a\}} f_j^2 \cdot Y_j^2\right] = \sum_{j\in[n]\setminus\{a\}} f_j^2 \cdot \mathbf{E}\left[Y_j^2\right]
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{E}\left[Z^2\right] &= \mathbf{E}\left[\sum_{j\in[n]\setminus\{a\}} f_j \cdot g(a)\cdot g(j)\,Y_j\right] \\
&= \mathbf{E}\left[\sum_{j\in[n]\setminus\{a\}} f_j^2 \cdot Y_j^2 + \sum_{j,j'\in[n]\setminus\{a\}:j\neq j;} f_j \cdot f_{j'} \cdot g(j)\cdot g(j')\cdot Y_j \cdot Y_{j'}\right] \\
&= \mathbf{E}\left[\sum_{j\in[n]\setminus\{a\}} f_j^2 \cdot Y_j^2\right] = \sum_{j\in[n]\setminus\{a\}} f_j^2 \cdot \mathbf{E}\left[Y_j^2\right]
\end{aligned}
$$

Note that for every $j \neq a$,

$$
\mathbf{E}\left[Y_j^2\right] = \mathbf{E}\left[Y_j\right] = \mathbf{Pr}\left[h(j) = h(a)\right] = \frac{1}{k}.
$$

$$
\begin{aligned}
\mathbf{E}\left[Z^2\right] &= \mathbf{E}\left[\sum_{j\in[n]\setminus\{a\}} f_j \cdot g(a)\cdot g(j)\, Y_j\right] \\
&= \mathbf{E}\left[\sum_{j\in[n]\setminus\{a\}} f_j^2 \cdot Y_j^2 + \sum_{j,j'\in[n]\setminus\{a\}:j\neq j;} f_j \cdot f_{j'} \cdot g(j)\cdot g(j')\cdot Y_j \cdot Y_{j'}\right] \\
&= \mathbf{E}\left[\sum_{j\in[n]\setminus\{a\}} f_j^2 \cdot Y_j^2\right] = \sum_{j\in[n]\setminus\{a\}} f_j^2 \cdot \mathbf{E}\left[Y_j^2\right]
\end{aligned}
$$

Note that for every $j \neq a$,

$$
\mathbf{E}\left[Y_j^2\right] = \mathbf{E}\left[Y_j\right] = \mathbf{Pr}\left[h(j) = h(a)\right] = \frac{1}{k}.
$$

Therefore

$$
\mathbf{E}\left[Z^2\right] = \frac{\sum_{j\in[n]\setminus\{a\}} f_j^2}{k} \leq \frac{\|\mathbf{f}\|_2^2}{k}.
$$

$$\mathbf{Var}\left[X\right] = \mathbf{Var}\left[Z\right] = \mathbf{E}\left[Z^2\right] - \left(\mathbf{E}\left[Z\right]\right)^2 \leq \frac{\|\mathbf{f}\|_2^2}{k}.$$

$$\mathbf{Var}[X] = \mathbf{Var}[Z] = \mathbf{E}\left[Z^2\right] - (\mathbf{E}[Z])^2 \le \frac{\|\mathbf{f}\|_2^2}{k}.$$

By Chebyshev,

$$\mathbf{Pr}\left[\left|\widehat{f}_a - f_a\right| \ge \varepsilon \|\mathbf{f}\|_2\right] \le \frac{1}{k\varepsilon^2} = \frac{1}{3}.$$

$$\mathbf{Var}[X] = \mathbf{Var}[Z] = \mathbf{E}\left[Z^2\right] - (\mathbf{E}[Z])^2 \le \frac{\|\mathbf{f}\|_2^2}{k}.$$

By Chebyshev,

$$\mathbf{Pr}\left[\left|\widehat{f}_a - f_a\right| \ge \varepsilon\|\mathbf{f}\|_2\right] \le \frac{1}{k\varepsilon^2} = \frac{1}{3}.$$

We can then use Median trick to boost the algorithm so that

- $\mathbf{Pr}\left[\left|\widehat{f}_a - f_a\right| \ge \varepsilon\|\mathbf{f}\|_2\right] \le \delta$;
- it costs $O\left(\frac{1}{\varepsilon^2}\log\frac{1}{\delta}(\log m + \log n)\right)$ bits of memeory.

$$\mathbf{Var}[X] = \mathbf{Var}[Z] = \mathbf{E}\left[Z^2\right] - (\mathbf{E}[Z])^2 \le \frac{\|\mathbf{f}\|_2^2}{k}.$$

By Chebyshev,

$$\mathbf{Pr}\left[\left|\widehat{f_a} - f_a\right| \ge \varepsilon\|\mathbf{f}\|_2\right] \le \frac{1}{k\varepsilon^2} = \frac{1}{3}.$$

We can then use Median trick to boost the algorithm so that

- $\mathbf{Pr}\left[\left|\widehat{f_a} - f_a\right| \ge \varepsilon\|\mathbf{f}\|_2\right] \le \delta$;
- it costs $O\left(\frac{1}{\varepsilon^2}\log\frac{1}{\delta}(\log m + \log n)\right)$ bits of memeory.

Compare the performance (in terms of accuracy and space consumption) of Misra-Gries and Count Sketch (Exercise).