Algorithms for Data Science: Lecture 4

Vasileios Nakos

National Technical University of Athens

April 2, 2021

Streaming algorithms

Algorithms for processing a stream of elements and maintaining statistics in *sublinear* space about the elements in the stream.

Streaming algorithms

Algorithms for processing a stream of elements and maintaining statistics in *sublinear* space about the elements in the stream.

Alternatively, low-space data structures for big data computation.

Algorithms for processing a stream of elements and maintaining statistics in *sublinear* space about the elements in the stream.

Alternatively, low-space data structures for big data computation.
$\star$ Compress data "on-the-fly": store a small piece of information sufficient to answer approximate answer for the data set.

Applications

- Traffic Monitoring in networks.

- Traffic Monitoring in networks.
- Very large Databases, i.e. estimating the size of a join.

Applications

- Traffic Monitoring in networks.
- Very large Databases, i.e. estimating the size of a join.
- Manipulation of astronomical (satellite imagery), financial data.

Applications

- Traffic Monitoring in networks.
- Very large Databases, i.e. estimating the size of a join.
- Manipulation of astronomical (satellite imagery), financial data.
- GPS or seismometer readings to detect geological anomalies, sensor networks etc

Applications

- Traffic Monitoring in networks.
- Very large Databases, i.e. estimating the size of a join.
- Manipulation of astronomical (satellite imagery), financial data.
- GPS or seismometer readings to detect geological anomalies, sensor networks etc
- Training Machine Learning models when the training data set is huge.

Input: A sequence of elements $x_1, x_2, \ldots, \subseteq [n]$, where you can think of n as 2^{64}.
Output: Number of *distinct* elements in x

Distinct Elements

Input: A sequence of elements $x_1, x_2, \ldots, \subseteq [n]$, where you can think of n as 2^{64}.
Output: Number of *distinct* elements in x

- Distinct users hitting a webpage.
- Distinct values in a column of a database
- Number of distinct queries to a search engine.
- Distinct patterns in DNA sequence.

Distinct Elements

Input: A sequence of elements $x_1, x_2, \ldots, \subseteq [n]$, where you can think of n as 2^{64}.
Output: Number of *distinct* elements in x

- Distinct users hitting a webpage.
- Distinct values in a column of a database
- Number of distinct queries to a search engine.
- Distinct patterns in DNA sequence.

Implementations used by Google (Sawzall, Dremel, PowerDrill), Yahoo, Twitter, Facebook Presto, etc.

Google
Big Query

F_2 moment estimation

Input: Vector $x \in \mathbb{R}^n$, updates (i, Δ) causing $x_i \leftarrow x_i + \Delta$.

Input: Vector $x \in \mathbb{R}^n$, updates (i, Δ) causing $x_i \leftarrow x_i + \Delta$.
Output: An approximate answer to $\sum_{i=1}^{n} x_i^2$.

- Anomaly detection in traffic monitoring.
- Detect DoS attacks.
- Database optimization engine to estimate *self join size*.
- Subroutine in many other streaming algorithms.

Architecture of a streaming algorithm

Implements two routines, **Update()** and **Query()** using space S usually *sublinear* in the size of the input, i.e. does **not** store the whole input.

Architecture of a streaming algorithm

Implements two routines, **Update()** and **Query()** using space S usually *sublinear* in the size of the input, i.e. does **not** store the whole input.

- Distinct elements: $O(\frac{\log\log n}{\epsilon^2} + \log n)$ bits of space to estimate number of distinct elements up to $(1+\epsilon)$
- F_2 estimation: $O(\frac{\log n}{\epsilon^2})$ bits of space to estimate the F_2 moment up to $1+\epsilon$, i.e. find V such that

$$(1-\epsilon)\sum_{i=1}^{n} x_i^2 \leq V \leq (1+\epsilon)\sum_{i=1}^{n} x_i^2 .$$

Streaming algorithms are almost always *randomized* and *approximate*.

Let X be a discrete random variable that takes values on $\{\ldots, -1, 0, 1 \ldots\}$. Then

- (expectation) $\mathbb{E}(X) := \sum_{i=-\infty}^{\infty} i \cdot Pr[X = i]$
- (variance) $\mathrm{Var}(X) := \mathbb{E}(X^2) - \mathbb{E}(X)^2$

Let X be a discrete random variable that takes values on $\{\ldots, -1, 0, 1 \ldots\}$. Then

- (expectation) $\mathbb{E}(X) := \sum_{i=-\infty}^{\infty} i \cdot Pr[X = i]$
- (variance) $\mathrm{Var}(X) := \mathbb{E}(X^2) - \mathbb{E}(X)^2$
- (linearity of expecation) $\mathbb{E}(X + Y) = \mathbb{E}(X) + \mathbb{E}(Y)$.

Let X be a discrete random variable that takes values on $\{\ldots, -1, 0, 1\ldots\}$. Then

- (expectation) $\mathbb{E}(X) := \sum_{i=-\infty}^{\infty} i \cdot Pr[X = i]$
- (variance) $\mathrm{Var}(X) := \mathbb{E}(X^2) - \mathbb{E}(X)^2$
- (linearity of expecation) $\mathbb{E}(X + Y) = \mathbb{E}(X) + \mathbb{E}(Y)$.
- (Markov's inequality) For a variable X that takes only positive values we have $Pr[X \geq x] \leq \frac{\mathbb{E}(X)}{x}$.

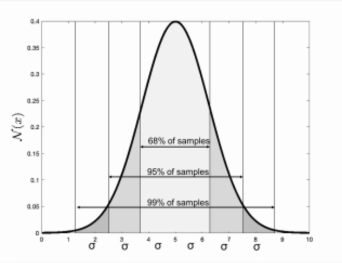
Let X be a discrete random variable that takes values on $\{\ldots, -1, 0, 1 \ldots\}$. Then

- (expectation) $\mathbb{E}(X) := \sum_{i=-\infty}^{\infty} i \cdot Pr[X = i]$
- (variance) $\mathrm{Var}(X) := \mathbb{E}(X^2) - \mathbb{E}(X)^2$
- (linearity of expecation) $\mathbb{E}(X + Y) = \mathbb{E}(X) + \mathbb{E}(Y)$.
- (Markov's inequality) For a variable X that takes only positive values we have $Pr[X \geq x] \leq \frac{\mathbb{E}(X)}{x}$.
- (Chebyshev's inequality) $Pr[|X - \mathbb{E}(X)| \geq \lambda] \leq \frac{\mathrm{Var}(X)}{\lambda^2}$.

Chebyshev's inequality is very useful in the design of randomized algorithms, showing that an estimator concentrates around its expected value.

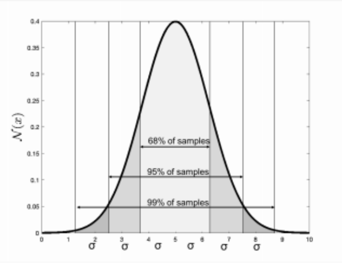
Question: How tight is Chebyshev's inequality?

Question: How tight is Chebyshev's inequality?



Follow the "68-95-99 rule" for Gaussian bell-curve $\mathcal{N}(0, \sigma^2)$.

Question: How tight is Chebyshev's inequality?



Follow the "68-95-99 rule" for Gaussian bell-curve $\mathcal{N}(0, \sigma^2)$.

- Chebyshev's inequality versus true value.
- $Pr[|X - \mathbb{E}(X)| \geq 1\sigma] \leq 100\%$ vs $Pr[|X - \mathbb{E}(X)| \geq 1\sigma] \approx 32\%$
- $Pr[|X - \mathbb{E}(X)| \geq 2\sigma] \leq 25\%$ vs $Pr[|X - \mathbb{E}(X)| \geq 1\sigma] \approx 5\%$
- $Pr[|X - \mathbb{E}(X)| \geq 3\sigma] \leq 11\%$ vs $Pr[|X - \mathbb{E}(X)| \geq 1\sigma] \approx 1\%$
- $Pr[|X - \mathbb{E}(X)| \geq 4\sigma] \leq 6\%$ vs $Pr[|X - \mathbb{E}(X)| \geq 1\sigma] \approx 0.01\%$

Input: A sequence of elements $x_1, x_2, \ldots, \subseteq [n]$, where you can think of n as 2^{64}.

Back to Distinct Elements

Input: A sequence of elements $x_1, x_2, \ldots, \subseteq [n]$, where you can think of n as 2^{64}.
Output: Number of *distinct* elements D in x. Example:
$0, 1, 0, 0, 0, 1, 2 \rightarrow 3$

Input: A sequence of elements $x_1, x_2, \ldots, \subseteq [n]$, where you can think of n as 2^{64}.
Output: Number of *distinct* elements D in x. Example:
$0, 1, 0, 0, 0, 1, 2 \rightarrow 3$

The (idealized) Flajolet-Martin algorithm:

- Choose a random hash function $h : [n] \rightarrow [0, 1]$
- $S = \infty$
- For every element e set $S \leftarrow \min\{S, h(e)\}$.
- Output $\frac{1}{S} - 1$.

Input: A sequence of elements $x_1, x_2, \ldots, \subseteq [n]$, where you can think of n as 2^{64}.
Output: Number of *distinct* elements D in x. Example:
$0, 1, 0, 0, 0, 1, 2 \to 3$

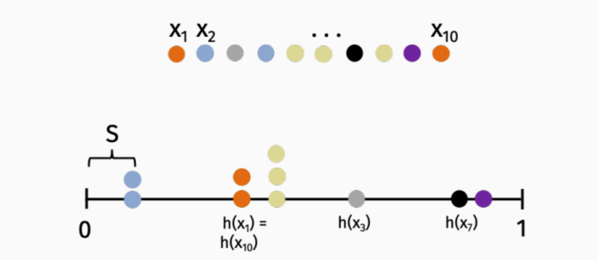
The (idealized) Flajolet-Martin algorithm:

- Choose a random hash function $h : [n] \to [0, 1]$
- $S = \infty$
- For every element e set $S \leftarrow \min\{S, h(e)\}$.
- Output $\frac{1}{S} - 1$.

We must store S and a description of $h \ldots$

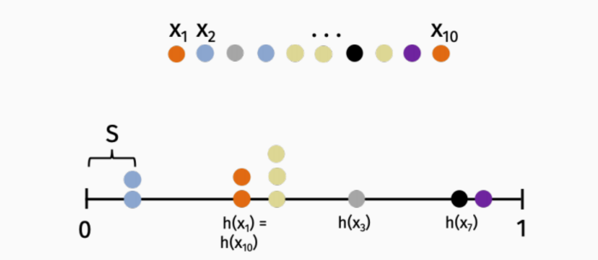
So what is V?

S is the minimum hash value ever seen so far.



So what is V?

S is the minimum hash value ever seen so far.



If an element comes many times, it will be always mapped to the same position. We return $\frac{1}{S} - 1$.

Lemma

$\mathbb{E}(S) = \frac{1}{D+1}$.

Lemma

$\mathbb{E}(S) = \frac{1}{D+1}$.

$$\mathbb{E}(S) = \int_0^1 Pr[S \geq \lambda] d\lambda =$$

$$\int_0^1 (1 - \lambda)^D d\lambda = \frac{1}{D+1}$$

We know that $\mathbb{E}(S) = \frac{1}{D+1}$. **Estimate** $\tilde{D} = \frac{1}{S} - 1$**.**

We know that $\mathbb{E}(S) = \frac{1}{D+1}$. **Estimate** $\tilde{D} = \frac{1}{S} - 1$.
Note that it **does not** hold that $\mathbb{E}(\tilde{D}) = D$.

We know that $\mathbb{E}(S) = \frac{1}{D+1}$. **Estimate** $\tilde{D} = \frac{1}{S} - 1$.
Note that it **does not** hold that $\mathbb{E}(\tilde{D}) = D$.

However,

$$|S - \mathbb{E}(S)| \leq \epsilon \cdot \mathbb{E}(S) \Rightarrow (1 - 4\epsilon)D \leq \widetilde{D} \leq (1 + 4\epsilon)D.$$

We know that $\mathbb{E}(S) = \frac{1}{D+1}$. **Estimate** $\tilde{D} = \frac{1}{S} - 1$.
Note that it **does not** hold that $\mathbb{E}(\tilde{D}) = D$.

However,

$$|S - \mathbb{E}(S)| \leq \epsilon \cdot \mathbb{E}(S) \Rightarrow (1 - 4\epsilon)D \leq \widetilde{D} \leq (1 + 4\epsilon)D.$$

Thus, precise estimation of D reduces to showing that S concentrates around its value $\rightarrow$ Chebyshev's inequality!

Let's compute the variance.

Lemma

$\mathrm{Var}(S) = \mathbb{E}(S^2) - \mathbb{E}(S)^2 = \frac{2}{(D+1)(D+2)} - \frac{1}{(D+1)^2} \leq \frac{1}{(D+1)^2}$.

Similarly as before,

$$\mathbb{E}(S^2) = \int_0^1 Pr[S^2 \geq \lambda] d\lambda =$$
$$\int_0^1 Pr[S \geq \sqrt{\lambda}] d\lambda =$$
$$\int_0^1 (1 - \sqrt{\lambda})^D d\lambda = \frac{2}{(D+1)(D+2)}$$

Let's see what we get

- $\mathbb{E}(S) = \frac{1}{D+1}$
- $\mathrm{Var}(S) \leq \frac{1}{(D+1)^2}$

Let's see what we get

- $\mathbb{E}(S) = \frac{1}{D+1}$
- $\mathrm{Var}(S) \leq \frac{1}{(D+1)^2}$
- $Pr[|S - \mathbb{E}(S)| \geq \epsilon\sqrt{\mathrm{Var}(S)}] \leq \frac{1}{\epsilon^2}$ (too large!)

Consider identicaly distributed random variables $S_1, S_2, \dots S_r$, and let

$$\bar{S} := \frac{1}{r}(S_1 + S_2 + \dots + S_r).$$

- $\mathbb{E}(S) = \mathbb{E}(\frac{1}{r}(S_1 + S_2 + \dots + S_r)) = \frac{1}{r}(\mathbb{E}(S)_1 + \mathbb{E}(S)_2 + \dots + \mathbb{E}(S)_r) = \mathbb{E}(S)$.

Consider identicaly distributed random variables $S_1, S_2, \ldots S_r$, and let

$$\bar{S} := \frac{1}{r}(S_1 + S_2 + \ldots + S_r).$$

- $\mathbb{E}(S) = \mathbb{E}(\frac{1}{r}(S_1 + S_2 + \ldots + S_r)) = \frac{1}{r}(\mathbb{E}(S)_1 + \mathbb{E}(S)_2 + \ldots + \mathbb{E}(S)_r) = \mathbb{E}(S)$.
- $\mathrm{Var}(S) = \frac{1}{r} \cdot \mathrm{Var}(S)$, since
    1. For random variables X, Y we have $\mathrm{Var}(X + Y) = \mathrm{Var}(X) + \mathrm{Var}(y)$, and

Consider identicaly distributed random variables $S_1, S_2, \ldots S_r$, and let

$$\bar{S} := \frac{1}{r}(S_1 + S_2 + \ldots + S_r).$$

- $\mathbb{E}(S) = \mathbb{E}(\frac{1}{r}(S_1 + S_2 + \ldots + S_r)) = \frac{1}{r}(\mathbb{E}(S)_1 + \mathbb{E}(S)_2 + \ldots + \mathbb{E}(S)_r) = \mathbb{E}(S)$.
- $\mathrm{Var}(S) = \frac{1}{r} \cdot \mathrm{Var}(S)$, since
  1. For random variables X, Y we have $\mathrm{Var}(X + Y) = \mathrm{Var}(X) + \mathrm{Var}(y)$, and
  2. $\mathrm{Var}(aX) = a^2\mathrm{Var}(X)$.

The boosting of independent repetitions

Consider identicaly distributed random variables $S_1, S_2, \ldots S_r$, and let

$$\bar{S} := \frac{1}{r}(S_1 + S_2 + \ldots + S_r).$$

- $\mathbb{E}(S) = \mathbb{E}(\frac{1}{r}(S_1 + S_2 + \ldots + S_r)) = \frac{1}{r}(\mathbb{E}(S)_1 + \mathbb{E}(S)_2 + \ldots + \mathbb{E}(S)_r) = \mathbb{E}(S)$.
- $\mathrm{Var}(S) = \frac{1}{r} \cdot \mathrm{Var}(S)$, since
    1. For random variables X, Y we have $\mathrm{Var}(X + Y) = \mathrm{Var}(X) + \mathrm{Var}(y)$, and
    2. $\mathrm{Var}(aX) = a^2\mathrm{Var}(X)$.

Thus, in our case take $r = O(\frac{1}{\epsilon^2})$ different instantiations of the algorithm, and output the average!

The (idealized) Flajolet-Martin algorithm:

- $r := \frac{3}{\epsilon^2}$
- Choose random hash function $h_r : [n] \to [0, 1]$ for all $j \in [r]$.
- For every $j \in [r]$ set $S_j = \infty$
- For every element e set and every $j \in [r]$ set $S_j \leftarrow \min \{S_j, h_j(e)\}$.
- $S := \frac{1}{r}(S_1 + S_2 + \ldots + S_r)$.
- Output $\frac{1}{S} - 1$.

The (idealized) Flajolet-Martin algorithm:

- $r := \frac{3}{\epsilon^2}$
- Choose random hash function $h_r : [n] \to [0, 1]$ for all $j \in [r]$.
- For every $j \in [r]$ set $S_j = \infty$
- For every element e set and every $j \in [r]$ set $S_j \leftarrow \min\{S_j, h_j(e)\}$.
- $S := \frac{1}{r}(S_1 + S_2 + \ldots + S_r)$.
- Output $\frac{1}{S} - 1$.

We have $\mathbb{E}(S) = \frac{1}{D+1}, \mathrm{Var}(S) < \frac{1}{(D+1)^2 r}$, so applying Chebyshev's inequality yields

$$Pr[|S - \mathbb{E}(S)| \geq \frac{\epsilon}{D+1}] \leq \frac{1}{3}.$$

Thus, with probability $\frac{2}{3}$ our estimator will satisfy

$$(1 - 4\epsilon)D \leq \widetilde{D} \leq (1 + 4\epsilon)D.$$

So we need to run the algorith with $\epsilon' := \frac{\epsilon}{4}$.

How to boost the success probability?

Idea I: For target probability δ, we can set $r = \frac{16}{\epsilon^2 \delta}$ and obtain an estimate satisfying

$$(1 - \epsilon)D \leq \widetilde{D} \leq (1 + \epsilon)D.$$

with probability $1 - \delta$.

How to boost the success probability?

Idea I: For target probability δ, we can set $r = \frac{16}{\epsilon^2\delta}$ and obtain an estimate satisfying

$$(1-\epsilon)D \leq \widetilde{D} \leq (1+\epsilon)D.$$

with probability $1-\delta$.
Why? $\mathrm{Var}(S)$ becomes $\frac{\epsilon^2\delta}{16\cdot(D+1)^2}$, so the same analysis yields

$$Pr[|S-\mathbb{E}(S)| \geq \frac{\epsilon}{4(D+1)}] \leq \delta.$$

But we can do waaay better!

For each $t \in [2\log(1/\delta)]$, keep a distinct elements data structures D_t with $O(\frac{1}{\epsilon^2})$ counters, and let $S^{(t)}$ be the estimate produced by each data structure.

For each $t \in [2\log(1/\delta)]$, keep a distinct elements data structures D_t with $O(\frac{1}{\epsilon^2})$ counters, and let $S^{(t)}$ be the estimate produced by each data structure.

Then the quantity

$$S := \mathrm{median}_t S^{(t)}$$

satisfies the desired inequality with probability $1 - \delta$. Good, huh?

We began with an *unbiased* estimator which has large variance, and

- for a multiplicative cost of $O(\frac{1}{\epsilon^2})$ we reduced the variance to the desired one (trick of the means).

What happened?

We began with an *unbiased* estimator which has large variance, and

- for a multiplicative cost of $O(\frac{1}{\epsilon^2})$ we reduced the variance to the desired one (trick of the means).
- for a multiplicative cost of $O(\log(1/\delta))$ we reduced the failure probability to δ (trick of the medians).

We began with an *unbiased* estimator which has large variance, and

- for a multiplicative cost of $O(\frac{1}{\epsilon^2})$ we reduced the variance to the desired one (trick of the means).
- for a multiplicative cost of $O(\log(1/\delta))$ we reduced the failure probability to δ (trick of the medians).

Space complexity (besides storing the hash functions):
$O(\log(1/\delta) \cdot \frac{1}{\epsilon^2} \cdot \log n)$ bits or $O(\log(1/\delta)/\epsilon^2)$ words.

We began with an *unbiased* estimator which has large variance, and

- for a multiplicative cost of $O(\frac{1}{\epsilon^2})$ we reduced the variance to the desired one (trick of the means).
- for a multiplicative cost of $O(\log(1/\delta))$ we reduced the failure probability to δ (trick of the medians).

Space complexity (besides storing the hash functions): $O(\log(1/\delta) \cdot \frac{1}{\epsilon^2} \cdot \log n)$ bits or $O(\log(1/\delta)/\epsilon^2)$ words. Compare with the trivial space of $O(D)$ words.

Avoiding taking $h : [n] \to [0, 1]$

$h(x_1)$	1010010
$h(x_2)$	1001100
$h(x_3)$	1001110
⋮	
$h(x_n)$	1011000

In the practical version of Flajolet-Martin (HyperLogLog) we estimate distinct elements based on maximum number of trailing zeros.

Avoiding taking $h : [n] \to [0, 1]$

$h(x_1)$	1010010
$h(x_2)$	1001100
$h(x_3)$	1001110
$\vdots$	
$h(x_n)$	1011000

In the practical version of Flajolet-Martin (HyperLogLog) we estimate distinct elements based on maximum number of trailing zeros.

$$\Pr[h(x) \text{ has } \log D \text{ trailing zeros}] = \frac{1}{D}.$$

What happens for the practical Flajolet-Martin?

Total space: $O(\log\log D/\epsilon^2 + \log D)$ for an ϵ approximation with constant probability.

Quote from "Loglog Counting of Large Cardinalities

"Using an auxiliary memory smaller than the size of this abstract, the LogLog algorithm makes it possible to estimate in a single pass and within a few percents the number of different words in the whole of Shakespeare's works." - Flajolet, Durand.

What happens for the practical Flajolet-Martin?

Total space: $O(\log\log D/\epsilon^2 + \log D)$ for an ϵ approximation with constant probability.

Quote from "Loglog Counting of Large Cardinalities

"Using an auxiliary memory smaller than the size of this abstract, the LogLog algorithm makes it possible to estimate in a single pass and within a few percents the number of different words in the whole of Shakespeare's works." - Flajolet, Durand.

Using HyperLogLog to approximate 1 billion distinct items to 2% accuracy can be in approximately $1.6\text{KB} = 12800$ bits!

Distinct Elements in the Distributed Setting

The Flajolet-Martin algorithm is totally distributed: why share lists of distinct elements when you can only share a bunch of minimum hash values seen?

The Flajolet-Martin algorithm is totally distributed: why share lists of distinct elements when you can only share a bunch of minimum hash values seen?

presto

Estimate spam rate: Count number of distinct subject lines in emails sent by users that have registered in the last week, in comparison to number of emails sent overall.

Distinct Elements in the Distributed Setting

The Flajolet-Martin algorithm is totally distributed: why share lists of distinct elements when you can only share a bunch of minimum hash values seen?

presto

Estimate spam rate: Count number of distinct subject lines in emails sent by users that have registered in the last week, in comparison to number of emails sent overall.

Good news: Answering the above query can be done in 2 seconds in Google's distributed implementations!

F_2 ESTIMATION

Input: A vector $x \in \mathbb{R}^n$ and updates (i, Δ) causing $x_i \leftarrow x_i + \Delta$.

Input: A vector $x \in \mathbb{R}^n$ and updates (i, Δ) causing $x_i \leftarrow x_i + \Delta$.
Output: A value V satisfying $(1 - \epsilon) \cdot \sum_{i=1}^n x_i^2 \leq V \leq (1 + \epsilon) \cdot \sum_{i=1}^n x_i^2$.

Alon-Mattias-Szegedy (AMS) sketch

There exists an algorithm which uses $O(\frac{\log n}{\epsilon^2})$ bits of space and returns an estimator as above with constant probability.

The algorithm and proof is just a couple of lines, yet the authors received the Goedel prize for that!

- $V \leftarrow 0$
- Pick hash function $\sigma : [n] \rightarrow \{-1, 1\}$ (random signs)

- $V \leftarrow 0$
- Pick hash function $\sigma : [n] \rightarrow \{-1, 1\}$ (random signs)
- Upon update (i, Δ) set $V \leftarrow V + \sigma(i) \cdot \Delta$

- $V \leftarrow 0$
- Pick hash function $\sigma : [n] \rightarrow \{-1, 1\}$ (random signs)
- Upon update (i, Δ) set $V \leftarrow V + \sigma(i) \cdot \Delta$
- Output V^2

- $V \leftarrow 0$
- Pick hash function $\sigma : [n] \to \{-1, 1\}$ (random signs)
- Upon update (i, Δ) set $V \leftarrow V + \sigma(i) \cdot \Delta$
- Output V^2

Note that $V = \sum_{i=1}^{n} \sigma(i) x_i$ and

- $V \leftarrow 0$
- Pick hash function $\sigma : [n] \to \{-1, 1\}$ (random signs)
- Upon update (i, Δ) set $V \leftarrow V + \sigma(i) \cdot \Delta$
- Output V^2

Note that $V = \sum_{i=1}^{n} \sigma(i) x_i$ and

$$V^2 = \sum_{i,j} \sigma(i)\sigma(j) x_i x_j.$$

Let $X := V^2$.

So what about $\mathbb{E}(X)$?

It holds that

$$
\begin{aligned}
\mathbb{E}(V^2) = \mathbb{E}(\sum_{i,j} \sigma(i)\sigma(j)x_i x_j) = \\
\sum_{i,j} \mathbb{E}(\sigma(i) \cdot \sigma(j)) \cdot x_i x_j = \\
\sum_{i \neq j} \mathbb{E}(\sigma(i)) \cdot \mathbb{E}(\sigma(j)) \cdot x_i x_j + \sum_i x_i^2 = \\
\sum_{i \neq j} 0 \cdot 0 \cdot x_i x_j + \sum_i x_i^2 = \sum_i x_i^2.
\end{aligned}
$$

And what about $\mathrm{Var}(X^2)$?

$$\mathrm{Var}(X) = \mathbb{E}(X^2) - (\mathbb{E}(X))^2 = \mathbb{E}(V^4) - (\mathbb{E}(V^2))^2 =$$

$$\mathbb{E}\left(\sum_{i_1,i_2,i_3,i_4} \sigma(i_1)\sigma(i_2)\sigma(i_3)\sigma(i_4)x_{i_1}x_{i_2}x_{i_3}x_{i_4}\right) =$$

$$\left(\sum_{i_1,i_2,i_3,i_4} \mathbb{E}(\sigma(i_1)\sigma(i_2)\sigma(i_3)\sigma(i_4))x_{i_1}x_{i_2}x_{i_3}x_{i_4}\right) =$$

$$\sum_{i=1}^{n} x_i^4 + \sum_{i,j} 6x_i^2x_j^2 \leq 3 \cdot \sum_{i=1}^{n} x_i^2 = 3\mathbb{E}(X).$$

And what about $\mathrm{Var}(X^2)$?

$$\mathrm{Var}(X) = \mathbb{E}(X^2) - (\mathbb{E}(X))^2 = \mathbb{E}(V^4) - (\mathbb{E}(V^2))^2 =$$

$$\mathbb{E}\left(\sum_{i_1,i_2,i_3,i_4} \sigma(i_1)\sigma(i_2)\sigma(i_3)\sigma(i_4)x_{i_1}x_{i_2}x_{i_3}x_{i_4}\right) =$$

$$\left(\sum_{i_1,i_2,i_3,i_4} \mathbb{E}(\sigma(i_1)\sigma(i_2)\sigma(i_3)\sigma(i_4))x_{i_1}x_{i_2}x_{i_3}x_{i_4}\right) =$$

$$\sum_{i=1}^{n} x_i^4 + \sum_{i,j} 6x_i^2x_j^2 \leq 3 \cdot \sum_{i=1}^{n} x_i^2 = 3\mathbb{E}(X).$$

If we apply Chebyshev's inequality, we run into the same issue as before (too large variance), so let's take $\frac{1}{\epsilon^2}$ different estimator and average them!

AMS SKETCH

- $r := \Theta(\frac{1}{\epsilon^2})$
- $V_j \leftarrow 0$ for all $j \in [r]$
- Pick hash functions $\sigma_j : [n] \rightarrow \{-1, 1\}$ (random signs) for all $j \in [n]$

- $r := \Theta(\frac{1}{\epsilon^2})$
- $V_j \leftarrow 0$ for all $j \in [r]$
- Pick hash functions $\sigma_j : [n] \to \{-1, 1\}$ (random signs) for all $j \in [n]$
- Upon update (i, Δ) set $V_j \leftarrow V_j + \sigma_j(i) \cdot \Delta$ for all $j \in [r]$

AMS sketch

- $r := \Theta(\frac{1}{\epsilon^2})$
- $V_j \leftarrow 0$ for all $j \in [r]$
- Pick hash functions $\sigma_j : [n] \to \{-1, 1\}$ (random signs) for all $j \in [n]$
- Upon update (i, Δ) set $V_j \leftarrow V_j + \sigma_j(i) \cdot \Delta$ for all $j \in [r]$
- Output $\frac{1}{r} \sum_{j=1}^{r} V_j^2$

- $r := \Theta(\frac{1}{\epsilon^2})$
- $V_j \leftarrow 0$ for all $j \in [r]$
- Pick hash functions $\sigma_j : [n] \rightarrow \{-1, 1\}$ (random signs) for all $j \in [n]$
- Upon update (i, Δ) set $V_j \leftarrow V_j + \sigma_j(i) \cdot \Delta$ for all $j \in [r]$
- Output $\frac{1}{r}\sum_{j=1}^{r} V_j^2$

To store each V_j we need $O(\log n)$ bits of space, for a total of all $O(\frac{\log n}{\epsilon^2})$ overall. Storing a compact representations of the hash functions can also be done in the same space.

- $r := \Theta(\frac{1}{\epsilon^2})$
- $V_j \leftarrow 0$ for all $j \in [r]$
- Pick hash functions $\sigma_j : [n] \to \{-1, 1\}$ (random signs) for all $j \in [n]$
- Upon update (i, Δ) set $V_j \leftarrow V_j + \sigma_j(i) \cdot \Delta$ for all $j \in [r]$
- Output $\frac{1}{r} \sum_{j=1}^{r} V_j^2$

To store each V_j we need $O(\log n)$ bits of space, for a total of all $O(\frac{\log n}{\epsilon^2})$ overall. Storing a compact representations of the hash functions can also be done in the same space.
Update time: $O(\frac{1}{\epsilon^2})$ assuming operations in happen constant time.
Query time: $O(\frac{1}{\epsilon^2})$ assuming operations happen in constant time.

What we've seen in this lecture.

- The abstract architecture of a streaming algorithm.
- Applications of streaming algorithms.
- Distinct elements and the power of randomness.
- Estimating the F_2 moment.

What we've seen in this lecture.

- The abstract architecture of a streaming algorithm.
- Applications of streaming algorithms.
- Distinct elements and the power of randomness.
- Estimating the F_2 moment.

Next lecture: Sketching and the foundations of Dimensionality Reduction.

What we've seen in this lecture.

- The abstract architecture of a streaming algorithm.
- Applications of streaming algorithms.
- Distinct elements and the power of randomness.
- Estimating the F_2 moment.

Next lecture: Sketching and the foundations of Dimensionality Reduction.
Thank you!