COMPSCI 514: ALGORITHMS FOR DATA SCIENCE

Cameron Musco

University of Massachusetts Amherst. Fall 2020.

Lecture 6

LOGISTICS

- Problem Set 1 is due tomorrow at 8pm in Gradescope.
- Quiz 3 will be due next Monday at 8pm on Moodle.

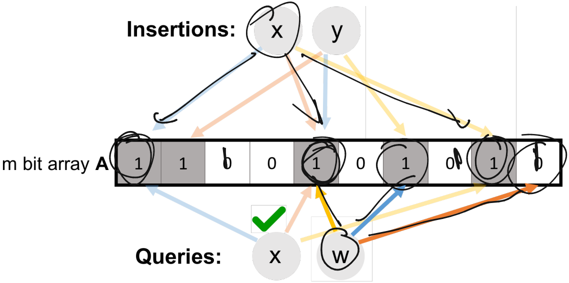
Last Class:

- Exponential concentration bound wrap up (central limit theorem, Chernoff bound).
- Bloom Filters:
  - Random hashing to maintain a large set in small space.
  - Discussed applications and how the false positive rate is determined.

This Class:

- Wrap up Bloom filters.
- Start on streaming algorithms – distinct items counting.

m-bit array. Each inserted item is marked with k bits, determined by k random hash functions.



- $query(x) = 1$ if and only if all bits that x hashes to are 1 (i.e., $A[h_1(x)] = \ldots = A[h_k(x)] = 1$.)
- Can be false positives, but no false negatives.

How does the false positive rate δ depend on m, k, and the number of items inserted n?

Step 1: What is the probability that after inserting n elements, the i^{th} bit of the array A is still 0?

$$\Pr(A[i] = 0) = \left(1 - \frac{1}{m}\right)^{kn} \approx e^{-\frac{kn}{m}}$$

n: total number items in filter, m: number of bits in filter, k: number of random hash functions, $\mathbf{h}_1, \ldots \mathbf{h}_k$: hash functions, A: bit array, δ: false positive rate.

How does the false positive rate δ depend on m, k, and the number of items inserted n?

Step 1: What is the probability that after inserting n elements, the i^{th} bit of the array A is still 0?

$$\Pr(A[i] = 0) = \left(1 - \frac{1}{m}\right)^{kn} \approx e^{-\frac{kn}{m}}$$

Step 2: What is the probability that querying a new item w gives a false positive?

n: total number items in filter, m: number of bits in filter, k: number of random hash functions, $\mathbf{h}_1, \ldots \mathbf{h}_k$: hash functions, A: bit array, δ: false positive rate.

How does the false positive rate δ depend on m, k, and the number of items inserted n?

Step 1: What is the probability that after inserting n elements, the i^{th} bit of the array A is still 0?

$$\Pr(A[i] = 0) = \left(1 - \frac{1}{m}\right)^{kn} \approx e^{-\frac{kn}{m}}$$

Step 2: What is the probability that querying a new item w gives a false positive?

$$\Pr\left(A[\mathbf{h}_1(w)] = \ldots = A[\mathbf{h}_k(w)] = 1\right)$$
$$= \Pr(A[\mathbf{h}_1(w)] = 1) \times \ldots \times \Pr(A[\mathbf{h}_k(w)] = 1)$$

n: total number items in filter, m: number of bits in filter, k: number of random hash functions, $\mathbf{h}_1, \ldots \mathbf{h}_k$: hash functions, A: bit array, δ: false positive rate.

How does the false positive rate δ depend on m, k, and the number of items inserted n?

Step 1: What is the probability that after inserting n elements, the i^{th} bit of the array A is still 0?

$$\Pr(A[i] = 0) = \left(1 - \frac{1}{m}\right)^{kn} \approx e^{-\frac{kn}{m}}$$

Step 2: What is the probability that querying a new item w gives a false positive?

$$\begin{aligned}\Pr\left(A[\mathbf{h}_1(w)] = \ldots = A[\mathbf{h}_k(w)] = 1\right) &= \Pr(A[\mathbf{h}_1(w)] = 1) \times \ldots \times \Pr(A[\mathbf{h}_k(w)] = 1) \\ &= \left(1 - e^{-\frac{kn}{m}}\right)^k\end{aligned}$$

Actually Incorrect! Dependent events.

> n: total number items in filter, m: number of bits in filter, k: number of random hash functions, $\mathbf{h}_1, \ldots \mathbf{h}_k$: hash functions, A: bit array, δ: false positive rate.

CORRECT ANALYSIS SKETCH

Step 1: To avoid dependence issues, condition on the event that the A has t zeros in it after n insertions, for some $t \leq m$. For a non-inserted element w, after conditioning on this event we correctly have:

$$\Pr(A[\mathbf{h}_1(w)] = \ldots = A[\mathbf{h}_k(w)] = 1)$$
$$= \Pr(A[\mathbf{h}_1(w)] = 1) \times \ldots \times \Pr(A[\mathbf{h}_k(w)] = 1).$$

I.e., the events $A[\mathbf{h}_1(w)] = 1, \ldots, A[\mathbf{h}_k(w)] = 1$ are independent conditioned on the number of bits set in A. Why?

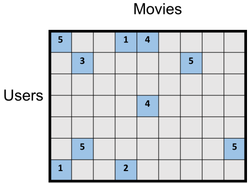
- Conditioned on this event, for any j, since $\mathbf{h}_j$ is a fully random hash function, $\Pr(A[\mathbf{h}_j(w)] = 1) = \frac{t}{m}$.
- Thus conditioned on this event, the false positive rate is $\left(1 - \frac{t}{m}\right)^k$.
- It remains to show that $\frac{t}{m} \approx e^{-\frac{kn}{m}}$ with high probability. We already have that $\mathbb{E}[\frac{t}{m}] = \frac{1}{m}\sum_{i=1}^{m} \Pr(A[i] = 0) \approx e^{-\frac{kn}{m}}$.

Need to show that the number of zeros t in A after n insertions is bounded by $O\left(e^{-\frac{kn}{m}}\right)$ with high probability.

Can apply Theorem 2 of: `http://cglab.ca/~morin/publications/ds/bloom-submitted.pdf`

False Positive Rate: with m bits of storage, k hash functions, and n items inserted $\delta \approx \left(1 - e^{\frac{-kn}{m}}\right)^k$.

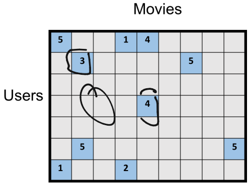
False Positive Rate: with m bits of storage, k hash functions, and n items inserted $\delta \approx \left(1 - e^{\frac{-kn}{m}}\right)^k$.



- We have 100 million users and $10,000$ movies. On average each user has rated only 10 movies so of these 10^{12} possible (user,movie) pairs, only $10 * 100,000,000 = 10^9 = n$ (user,movie) pairs have non-empty entries in our table.
- We allocate $m = 8n = 8 \times 10^9$ bits for a Bloom filter (1 GB).

FALSE POSITIVE RATE

False Positive Rate: with m bits of storage, k hash functions, and n items inserted $\delta \approx \left(1 - e^{\frac{-kn}{m}}\right)^k$.

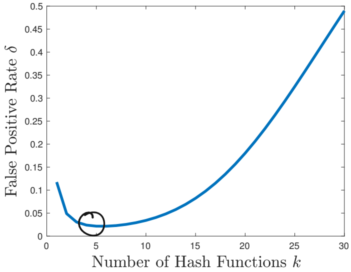


- We have 100 million users and $10,000$ movies. On average each user has rated only 10 movies so of these 10^{12} possible (user,movie) pairs, only $10 * 100,000,000 = 10^9 = n$ (user,movie) pairs have non-empty entries in our table.
- We allocate $m = 8n = 8 \times 10^9$ bits for a Bloom filter (1 GB). How should we set k to minimize the number of false positives?

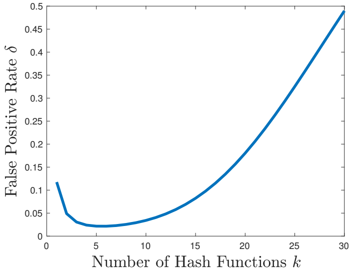
False Positive Rate: with m bits of storage, k hash functions, and n items inserted $\delta \approx \left(1 - e^{\frac{-kn}{m}}\right)^{k}$.

$n = 10^9$

$m = 8 \cdot 10^9$



False Positive Rate: with m bits of storage, k hash functions, and n items inserted $\delta \approx \left(1 - e^{\frac{-kn}{m}}\right)^k$.



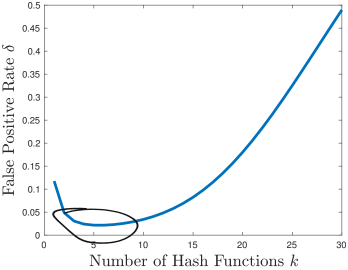
- Can differentiate to show optimal number of hashes is $k = \ln 2 \cdot \frac{m}{n}$.

False Positive Rate: with m bits of storage, k hash functions, and n items inserted $\delta \approx \left(1 - e^{\frac{-kn}{m}}\right)^k$.

(fraction of 1 entries)^k

$1 - e^{-\frac{\ln 2 \cdot \frac{m}{n} \cdot n}{m}}$

$1 - e^{-\ln 2} = \frac{1}{2}$



- Can differentiate to show optimal number of hashes is $k = \ln 2 \cdot \frac{m}{n}$.
- Balances between filling up the array with too many hashes and having enough hashes so that even when the array is pretty full, a new item is unlikely to have all its bits set (yield a false positive)

An observation about Bloom filter space complexity: m?

An observation about Bloom filter space complexity:

$$\text{False Positive Rate: } \delta \approx \left(1 - e^{-\frac{kn}{m}}\right)^k.$$

For an m-bit bloom filter holding n items, optimal number of hash functions k is: $k = \ln 2 \cdot \frac{m}{n}$.

An observation about Bloom filter space complexity:

$$\text{False Positive Rate: } \delta \approx \left(1 - e^{-\frac{kn}{m}}\right)^k.$$

For an m-bit bloom filter holding n items, optimal number of hash functions k is: $k = \ln 2 \cdot \frac{m}{n}$.

If we want a false positive rate $< \frac{1}{2}$ how big does m need to be in comparison to n?

$$m = O(\log n),\ m = O(\sqrt{n}),\ m = O(n),\ m = O(n^2)?$$

An observation about Bloom filter space complexity:

$$\text{False Positive Rate: } \delta \approx \left(1 - e^{-\frac{kn}{m}}\right)^k.$$

For an m-bit bloom filter holding n items, optimal number of hash functions k is: $k = \ln 2 \cdot \frac{m}{n}$.

If we want a false positive rate $< \frac{1}{2}$ how big does m need to be in comparison to n?

$$m = O(\log n), \; m = O(\sqrt{n}), \; m = O(n), \; m = O(n^2)?$$

An observation about Bloom filter space complexity:

$$\text{False Positive Rate: } \delta \approx \left(1 - e^{-\frac{kn}{m}}\right)^k.$$

For an m-bit bloom filter holding n items, optimal number of hash functions k is: $k = \ln 2 \cdot \frac{m}{n}$.

If we want a false positive rate $< \frac{1}{2}$ how big does m need to be in comparison to n?

$$m = O(\log n), \;\; m = O(\sqrt{n}), \;\; m = O(n), \;\; m = O(n^2)?$$

If $m = \frac{n}{\ln 2}$, optimal $k = 1$, and failure rate is:

$$\delta = \left(1 - e^{-\frac{n/\ln 2}{n}}\right)^1 = \left(1 - \frac{1}{2}\right)^1 = \frac{1}{2}.$$

An observation about Bloom filter space complexity:

$$\text{False Positive Rate: } \delta \approx \left(1 - e^{-\frac{kn}{m}}\right)^k.$$

For an m-bit bloom filter holding n items, optimal number of hash functions k is: $k = \ln 2 \cdot \frac{m}{n}$.

If we want a false positive rate $< \frac{1}{2}$ how big does m need to be in comparison to n?

$$m = O(\log n),\ m = O(\sqrt{n}),\ m = O(n),\ m = O(n^2)?$$

If $m = \frac{n}{\ln 2}$, optimal $k = 1$, and failure rate is:

$$\delta = \left(1 - e^{-\frac{n/\ln 2}{n}}\right)^1 = \left(1 - \frac{1}{2}\right)^1 = \frac{1}{2}.$$

I.e., storing n items in a bloom filter requires $O(n)$ space. So what's the point?

An observation about Bloom filter space complexity:

$$\text{False Positive Rate: } \delta \approx \left(1 - e^{-\frac{kn}{m}}\right)^k.$$

For an m-bit bloom filter holding n items, optimal number of hash functions k is: $k = \ln 2 \cdot \frac{m}{n}$.

If we want a false positive rate $< \frac{1}{2}$ how big does m need to be in comparison to n?

$$m = O(\log n),\ m = O(\sqrt{n}),\ m = O(n),\ m = O(n^2)?$$

If $m = \frac{n}{\ln 2}$, optimal $k = 1$, and failure rate is:

$$\delta = \left(1 - e^{-\frac{n/\ln 2}{n}}\right)^1 = \left(1 - \frac{1}{2}\right)^1 = \frac{1}{2}.$$

I.e., storing n items in a bloom filter requires $O(n)$ space. So what's the point? Truly $O(n)$ bits, rather than $O(n \cdot \text{item size})$.

Questions on Bloom Filters?

Stream Processing: Have a massive dataset X with n items $x_1, x_2, \ldots, x_n$ that arrive in a continuous stream. Not nearly enough space to store all the items (in a single location).

- Still want to analyze and learn from this data.

Stream Processing: Have a massive dataset X with n items $x_1, x_2, \ldots, x_n$ that arrive in a continuous stream. Not nearly enough space to store all the items (in a single location).

- Still want to analyze and learn from this data.
- Typically must compress the data on the fly, storing a data structure from which you can still learn useful information.

Stream Processing: Have a massive dataset X with n items $x_1, x_2, \ldots, x_n$ that arrive in a continuous stream. Not nearly enough space to store all the items (in a single location).

- Still want to analyze and learn from this data.
- Typically must compress the data on the fly, storing a data structure from which you can still learn useful information.
- Often the compression is randomized. E.g., bloom filters.

Stream Processing: Have a massive dataset X with n items $x_1, x_2, \ldots, x_n$ that arrive in a continuous stream. Not nearly enough space to store all the items (in a single location).

- Still want to analyze and learn from this data.
- Typically must compress the data on the fly, storing a data structure from which you can still learn useful information.
- Often the compression is randomized. E.g., bloom filters.
- Compared to traditional algorithm design, which focuses on minimizing runtime, the big question here is how much space is needed to answer queries of interest.

- **Sensor data:** images from telescopes (15 terabytes per night from the Large Synoptic Survey Telescope), readings from seismometer arrays monitoring and predicting earthquake activity, traffic cameras and travel time sensors (Smart Cities), electrical grid monitoring.

- **Sensor data:** images from telescopes (15 terabytes per night from the Large Synoptic Survey Telescope), readings from seismometer arrays monitoring and predicting earthquake activity, traffic cameras and travel time sensors (Smart Cities), electrical grid monitoring.

SOME EXAMPLES

- **Sensor data:** images from telescopes (15 terabytes per night from the Large Synoptic Survey Telescope), readings from seismometer arrays monitoring and predicting earthquake activity, traffic cameras and travel time sensors (Smart Cities), electrical grid monitoring.
- **Internet Traffic**: 500 million Tweets per day, 5.6 billion Google searches, billions of ad-clicks and other logs from instrumented webpages, IPs routed by network switches, ...

SOME EXAMPLES

- Sensor data: images from telescopes (15 terabytes per night from the Large Synoptic Survey Telescope), readings from seismometer arrays monitoring and predicting earthquake activity, traffic cameras and travel time sensors (Smart Cities), electrical grid monitoring.
- Internet Traffic: 500 million Tweets per day, 5.6 billion Google searches, billions of ad-clicks and other logs from instrumented webpages, IPs routed by network switches, ...
- Datasets in Machine Learning: When training e.g. a neural network on a large dataset (ImageNet with 14 million images), the data is typically processed in a stream due to storage limitations.

SOME EXAMPLES

- **Sensor data:** images from telescopes (15 terabytes per night from the Large Synoptic Survey Telescope), readings from seismometer arrays monitoring and predicting earthquake activity, traffic cameras and travel time sensors (Smart Cities), electrical grid monitoring.
- **Internet Traffic**: 500 million Tweets per day, 5.6 billion Google searches, billions of ad-clicks and other logs from instrumented webpages, IPs routed by network switches, ...
- **Datasets in Machine Learning:** When training e.g. a neural network on a large dataset (ImageNet with 14 million images), the data is typically processed in a stream due to storage limitations.

Distinct Elements (Count-Distinct) Problem: Given a stream $x_1, \ldots, x_n$, output the number of distinct elements in the stream.

Distinct Elements (Count-Distinct) Problem: Given a stream $x_1, \ldots, x_n$, output the number of distinct elements in the stream. E.g.,

$$1, 5, 7, 5, 2, 1 \rightarrow 4 \text{ distinct elements}$$

Distinct Elements (Count-Distinct) Problem: Given a stream $x_1, \ldots, x_n$, estimate the number of distinct elements in the stream. E.g.,

$$1, 5, 7, 5, 2, 1 \rightarrow 4 \text{ distinct elements}$$

Distinct Elements (Count-Distinct) Problem: Given a stream $x_1, \ldots, x_n$, estimate the number of distinct elements in the stream. E.g.,

$$1, 5, 7, 5, 2, 1 \rightarrow 4 \text{ distinct elements}$$

Applications:

- Distinct IP addresses clicking on an ad or visiting a site.
- Distinct values in a database column (for estimating sizes of joins and group bys).
- Number of distinct search engine queries.
- Counting distinct motifs in large DNA sequences.

Distinct Elements (Count-Distinct) Problem: Given a stream $x_1, \ldots, x_n$, estimate the number of distinct elements in the stream. E.g.,

$$1, 5, 7, 5, 2, 1 \rightarrow 4 \text{ distinct elements}$$

Applications:

- Distinct IP addresses clicking on an ad or visiting a site.
- Distinct values in a database column (for estimating sizes of joins and group bys).
- Number of distinct search engine queries.
- Counting distinct motifs in large DNA sequences.

Google Sawzall, Facebook Presto, Apache Drill, Twitter Algebird

Distinct Elements (Count-Distinct) Problem: Given a stream $x_1, \ldots, x_n$, estimate the number of distinct elements in the stream. E.g.,

$$1, 5, 7, 5, 2, 1 \rightarrow 4 \text{ distinct elements}$$

Applications:

- Distinct IP addresses clicking on an ad or visiting a site.
- Distinct values in a database column (for estimating sizes of joins and group bys).
- Number of distinct search engine queries.
- Counting distinct motifs in large DNA sequences.

Google Sawzall, Facebook Presto, Apache Drill, Twitter Algebird

Breakout Rooms: Discuss ways you might solve this problem without storing the full list of items seen.

Bloom filter: insert every element as it comes in

O(d)

~~positive~~ negative query → increase your count

Hash functions: use multiple hash functions
more distinct elements → more hashed values

(somewhat similar

Distinct Elements (Count-Distinct) Problem: Given a stream $x_1, \ldots, x_n$, estimate the number of distinct elements.

Distinct Elements (Count-Distinct) Problem: Given a stream $x_1, \ldots, x_n$, estimate the number of distinct elements.

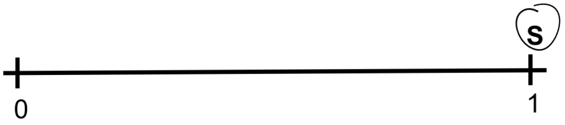
Min-Hashing for Distinct Elements (variant of Flajolet-Martin):

- Let $\mathbf{h} : U \to [0, 1]$ be a random hash function (with a real valued output)
- $s := 1$
- For $i = 1, \ldots, n$
  - $s := \min(s, \mathbf{h}(x_i))$
- Return $\tilde{d} = \frac{1}{s} - 1$

Distinct Elements (Count-Distinct) Problem: Given a stream $x_1, \ldots, x_n$, estimate the number of distinct elements.

Min-Hashing for Distinct Elements (variant of Flajolet-Martin):

- Let $\mathbf{h} : U \to [0, 1]$ be a random hash function (with a real valued output)
- $s := 1$
- For $i = 1, \ldots, n$
  - $s := \min(s, \mathbf{h}(x_i))$
- Return $\tilde{d} = \frac{1}{s} - 1$



Distinct Elements (Count-Distinct) Problem: Given a stream $x_1, \ldots, x_n$, estimate the number of distinct elements.

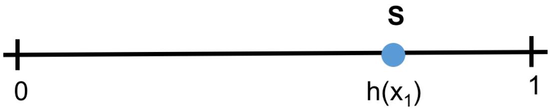
Min-Hashing for Distinct Elements (variant of Flajolet-Martin):

- Let $\mathbf{h} : U \to [0, 1]$ be a random hash function (with a real valued output)
- $s := 1$
- For $i = 1, \ldots, n$
  - $s := \min(s, \mathbf{h}(x_i))$
- Return $\tilde{d} = \frac{1}{s} - 1$

Distinct Elements (Count-Distinct) Problem: Given a stream $x_1, \ldots, x_n$, estimate the number of distinct elements.

Min-Hashing for Distinct Elements (variant of Flajolet-Martin):

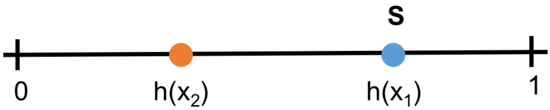
- Let $\mathbf{h} : U \to [0, 1]$ be a random hash function (with a real valued output)
- $s := 1$
- For $i = 1, \ldots, n$
  - $s := \min(s, \mathbf{h}(x_i))$
- Return $\tilde{d} = \frac{1}{s} - 1$



Distinct Elements (Count-Distinct) Problem: Given a stream $x_1, \ldots, x_n$, estimate the number of distinct elements.

Min-Hashing for Distinct Elements (variant of Flajolet-Martin):

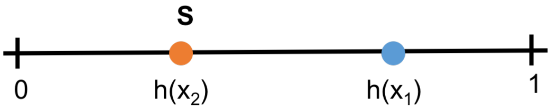
- Let $\mathbf{h} : U \to [0, 1]$ be a random hash function (with a real valued output)
- $s := 1$
- For $i = 1, \ldots, n$
  - $s := \min(s, \mathbf{h}(x_i))$
- Return $\tilde{d} = \frac{1}{s} - 1$



Distinct Elements (Count-Distinct) Problem: Given a stream $x_1, \ldots, x_n$, estimate the number of distinct elements.

Min-Hashing for Distinct Elements (variant of Flajolet-Martin):

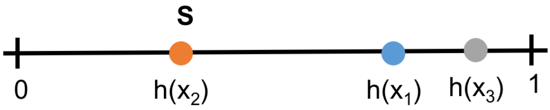
- Let $\mathbf{h} : U \to [0, 1]$ be a random hash function (with a real valued output)
- $s := 1$
- For $i = 1, \ldots, n$
  - $s := \min(s, \mathbf{h}(x_i))$
- Return $\tilde{d} = \frac{1}{s} - 1$



Distinct Elements (Count-Distinct) Problem: Given a stream $x_1, \ldots, x_n$, estimate the number of distinct elements.

Min-Hashing for Distinct Elements (variant of Flajolet-Martin):

- Let $\mathbf{h} : U \to [0, 1]$ be a random hash function (with a real valued output)
- $s := 1$
- For $i = 1, \ldots, n$
    - $s := \min(s, \mathbf{h}(x_i))$
- Return $\tilde{d} = \frac{1}{s} - 1$

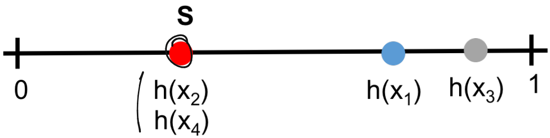


Distinct Elements (Count-Distinct) Problem: Given a stream $x_1, \ldots, x_n$, estimate the number of distinct elements.

Min-Hashing for Distinct Elements (variant of Flajolet-Martin):

- Let $\mathbf{h} : U \rightarrow [0, 1]$ be a random hash function (with a real valued output)
- $s := 1$
- For $i = 1, \ldots, n$
  - $s := \min(s, \mathbf{h}(x_i))$
- Return $\tilde{d} = \frac{1}{s} - 1$

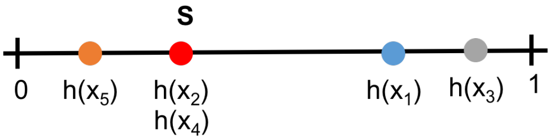
$x_2 = x_4$



Distinct Elements (Count-Distinct) Problem: Given a stream $x_1, \ldots, x_n$, estimate the number of distinct elements.

Min-Hashing for Distinct Elements (variant of Flajolet-Martin):

- Let $\mathbf{h} : U \to [0, 1]$ be a random hash function (with a real valued output)
- $s := 1$
- For $i = 1, \ldots, n$
  - $s := \min(s, \mathbf{h}(x_i))$
- Return $\tilde{d} = \frac{1}{s} - 1$



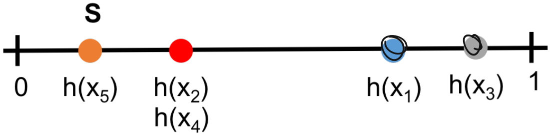
5 6 7 5 5 1 5

Distinct Elements (Count-Distinct) Problem: Given a stream $x_1, \ldots, x_n$, estimate the number of distinct elements.

Min-Hashing for Distinct Elements (variant of Flajolet-Martin):

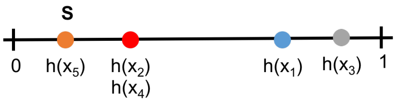
- Let $\mathbf{h} : U \to [0, 1]$ be a random hash function (with a real valued output)
- $s := 1$
- For $i = 1, \ldots, n$
  - $s := \min(s, \mathbf{h}(x_i))$
- Return $\tilde{d} = \frac{1}{s} - 1$

if # distinct items in large
s should be small



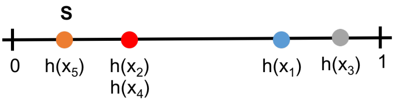
Min-Hashing for Distinct Elements:

- Let $\mathbf{h} : U \to [0, 1]$ be a random hash function (with a real valued output)
- $s := 1$
- For $i = 1, \ldots, n$
  - $s := \min(s, \mathbf{h}(x_i))$
- Return $\tilde{d} = \frac{1}{s} - 1$



Min-Hashing for Distinct Elements:

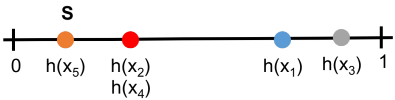
- Let $\mathbf{h} : U \to [0, 1]$ be a random hash function (with a real valued output)
- $s := 1$
- For $i = 1, \ldots, n$
  - $s := \min(s, \mathbf{h}(x_i))$
- Return $\tilde{d} = \frac{1}{s} - 1$



- After all items are processed, s is the minimum of d points chosen uniformly at random on $[0, 1]$. Where $d = \#$ distinct elements.

Min-Hashing for Distinct Elements:

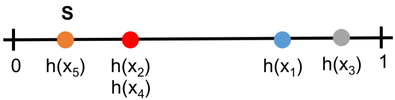
- Let $\mathbf{h} : U \rightarrow [0, 1]$ be a random hash function (with a real valued output)
- $s := 1$
- For $i = 1, \ldots, n$
  - $s := \min(s, \mathbf{h}(x_i))$
- Return $\tilde{d} = \frac{1}{s} - 1$



- After all items are processed, s is the minimum of d points chosen uniformly at random on $[0, 1]$. Where $d = \#$ distinct elements.
- Intuition: The larger d is, the smaller we expect s to be.

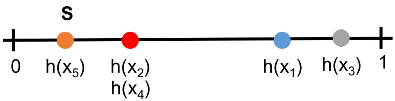
Min-Hashing for Distinct Elements:

- Let $\mathbf{h} : U \to [0, 1]$ be a random hash function (with a real valued output)
- $s := 1$
- For $i = 1, \ldots, n$
  - $s := \min(s, \mathbf{h}(x_i))$
- Return $\tilde{d} = \frac{1}{s} - 1$



- After all items are processed, s is the minimum of d points chosen uniformly at random on $[0, 1]$. Where $d = \#$ distinct elements.
- Intuition: The larger d is, the smaller we expect s to be.
- Same idea as Flajolet-Martin algorithm and HyperLogLog, except they use discrete hash functions.

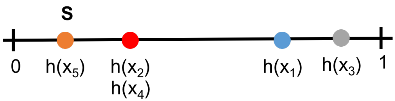
s is the minimum of d points chosen uniformly at random on $[0, 1]$.
Where $d = \#$ distinct elements.



PERFORMANCE IN EXPECTATION

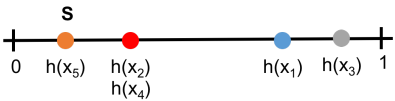
s is the minimum of d points chosen uniformly at random on $[0, 1]$.
Where $d = \#$ distinct elements.

$\hat{d} = \frac{1}{s} - 1$



$\mathbb{E}[s] =$

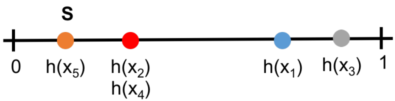
s is the minimum of d points chosen uniformly at random on $[0, 1]$.
Where $d = \#$ distinct elements.



$$\mathbb{E}[\mathbf{s}] = \frac{1}{d+1} \text{ (using } \mathbb{E}(\mathbf{s}) = \int_0^\infty \Pr(\mathbf{s} > x)dx\text{) + calculus)}$$

PERFORMANCE IN EXPECTATION

s is the minimum of d points chosen uniformly at random on $[0, 1]$. Where $d = \#$ distinct elements.



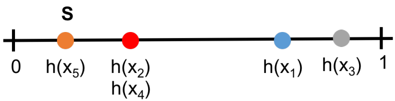
$$\mathbb{E}[\mathbf{s}] = \frac{1}{d+1} \text{ (using } \mathbb{E}(\mathbf{s}) = \int_0^\infty \Pr(\mathbf{s} > x) dx) + \text{calculus)}$$

- So estimate of $\widehat{\mathbf{d}} = \frac{1}{\mathbf{s}} - 1$ output by the algorithm is correct if **s** exactly equals its expectation.

$\frac{1}{1/d+1} - 1 = d$

PERFORMANCE IN EXPECTATION

s is the minimum of d points chosen uniformly at random on $[0, 1]$. Where $d = \#$ distinct elements.



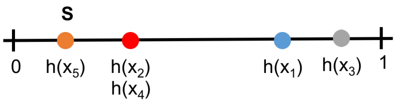
$$\mathbb{E}[\mathbf{s}] = \frac{1}{d+1} \text{ (using } \mathbb{E}(\mathbf{s}) = \int_0^\infty \Pr(\mathbf{s} > x)dx) + \text{calculus)}$$

$\mathbb{E}s = \frac{1}{d+1}$ $\frac{1}{\mathbb{E}[s]} - 1 = d$

- So estimate of $\widehat{\mathbf{d}} = \frac{1}{\mathbf{s}} - 1$ output by the algorithm is correct if **s** exactly equals its expectation. Does this mean $\mathbb{E}[\widehat{\mathbf{d}}] = d$?

$\mathbb{E}\hat{d} = \frac{1}{\mathbb{E}s} - 1 = d$

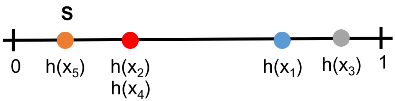
$\mathbf{s}$ is the minimum of d points chosen uniformly at random on $[0, 1]$. Where $d = \#$ distinct elements.



$$\mathbb{E}[\mathbf{s}] = \frac{1}{d+1} \text{ (using } \mathbb{E}(\mathbf{s}) = \int_0^\infty \Pr(\mathbf{s} > x)dx) + \text{calculus)}$$

- So estimate of $\widehat{\mathbf{d}} = \frac{1}{\mathbf{s}} - 1$ output by the algorithm is correct if $\mathbf{s}$ exactly equals its expectation. Does this mean $\mathbb{E}[\widehat{\mathbf{d}}] = d$? No, but:

s is the minimum of d points chosen uniformly at random on $[0,1]$.
Where $d = \#$ distinct elements.



$$\mathbb{E}[\mathbf{s}] = \frac{1}{d+1} \text{ (using } \mathbb{E}(\mathbf{s}) = \int_0^\infty \Pr(\mathbf{s} > x)dx\text{) + calculus)}$$

$\frac{1}{d+1}$

- So estimate of $\widehat{\mathbf{d}} = \frac{1}{\mathbf{s}} - 1$ output by the algorithm is correct if **s** exactly equals its expectation. Does this mean $\mathbb{E}[\widehat{\mathbf{d}}] = d$? No, but:
- **Approximation is robust:** if $|\mathbf{s} - \mathbb{E}[\mathbf{s}]| \leq \epsilon \cdot \mathbb{E}[\mathbf{s}]$ for any $\epsilon \in (0, 1/2)$ and a small constant $c \leq 4$:

$$(1 - c\epsilon)d \leq \widehat{\mathbf{d}} \leq (1 + c\epsilon)d$$

So question is how well s concentrates around its mean.

$$\mathbb{E}[s] = \frac{1}{d+1} \quad .$$

$$.$$

s: minimum of d distinct hashes chosen randomly over $[0, 1]$, computed by hashing algorithm. $\widehat{d} = \frac{1}{s} - 1$: estimate of # distinct elements d.

So question is how well **s** concentrates around its mean.

$$\mathbb{E}[\mathbf{s}] = \frac{1}{d+1} \text{ and } \mathrm{Var}[\mathbf{s}] \leq \frac{1}{(d+1)^2} \text{ (\textit{also via calculus})}.$$

.

s: minimum of d distinct hashes chosen randomly over $[0, 1]$, computed by hashing algorithm. $\widehat{\mathbf{d}} = \frac{1}{\mathbf{s}} - 1$: estimate of # distinct elements d.

INITIAL CONCENTRATION BOUND

So question is how well s concentrates around its mean.

$$\mathbb{E}[\mathsf{s}] = \frac{1}{d+1} \text{ and } \mathrm{Var}[\mathsf{s}] \leq \frac{1}{(d+1)^2} \text{ (\textit{also via calculus})}.$$

Chebyshev's Inequality:

$$\Pr\left[|\mathsf{s} - \mathbb{E}[\mathsf{s}]| \geq \epsilon \mathbb{E}[\mathsf{s}]\right] \leq \frac{\mathrm{Var}[\mathsf{s}]}{(\epsilon \mathbb{E}[\mathsf{s}])^2} = \frac{\frac{1}{(d+1)^2}}{\epsilon^2 \cdot \frac{1}{(d+1)^2}} = \frac{1}{\epsilon^2}$$

s: minimum of d distinct hashes chosen randomly over $[0,1]$, computed by hashing algorithm. $\widehat{\mathsf{d}} = \frac{1}{\mathsf{s}} - 1$: estimate of # distinct elements d.

So question is how well s concentrates around its mean.

$$\mathbb{E}[\mathsf{s}] = \frac{1}{d+1} \text{ and } \operatorname{Var}[\mathsf{s}] \leq \frac{1}{(d+1)^2} \text{ (*also via calculus*).}$$

Chebyshev's Inequality:

$$\Pr\left[|\mathsf{s} - \mathbb{E}[\mathsf{s}]| \geq \epsilon \mathbb{E}[\mathsf{s}]\right] \leq \frac{\operatorname{Var}[\mathsf{s}]}{(\epsilon \mathbb{E}[\mathsf{s}])^2} = \frac{1}{\epsilon^2}.$$

s: minimum of d distinct hashes chosen randomly over $[0, 1]$, computed by hashing algorithm. $\widehat{\mathsf{d}} = \frac{1}{\mathsf{s}} - 1$: estimate of # distinct elements d.

So question is how well s concentrates around its mean.

$$\mathbb{E}[\mathsf{s}] = \frac{1}{d+1} \text{ and } \mathrm{Var}[\mathsf{s}] \leq \frac{1}{(d+1)^2} \text{ (\textit{also via calculus})}.$$

Chebyshev's Inequality:

$$\Pr\left[|\mathsf{s} - \mathbb{E}[\mathsf{s}]| \geq \epsilon\mathbb{E}[\mathsf{s}]\right] \leq \frac{\mathrm{Var}[\mathsf{s}]}{(\epsilon\mathbb{E}[\mathsf{s}])^2} = \frac{1}{\epsilon^2}.$$

Bound is vacuous for any $\epsilon < 1$.

s: minimum of d distinct hashes chosen randomly over $[0,1]$, computed by hashing algorithm. $\widehat{\mathsf{d}} = \frac{1}{\mathsf{s}} - 1$: estimate of # distinct elements d.

So question is how well $\mathbf{s}$ concentrates around its mean.

$$\mathbb{E}[\mathbf{s}] = \frac{1}{d+1} \text{ and } \mathrm{Var}[\mathbf{s}] \leq \frac{1}{(d+1)^2} \text{ (\textit{also via calculus})}.$$

Chebyshev's Inequality:

$$\Pr\left[|\mathbf{s} - \mathbb{E}[\mathbf{s}]| \geq \epsilon \mathbb{E}[\mathbf{s}]\right] \leq \frac{\mathrm{Var}[\mathbf{s}]}{(\epsilon \mathbb{E}[\mathbf{s}])^2} = \frac{1}{\epsilon^2}.$$

Bound is vacuous for any $\epsilon < 1$. How can we improve accuracy?

$\mathbf{s}$: minimum of d distinct hashes chosen randomly over $[0, 1]$, computed by hashing algorithm. $\widehat{\mathbf{d}} = \frac{1}{\mathbf{s}} - 1$: estimate of # distinct elements d.

Leverage the law of large numbers: improve accuracy via repeated independent trials.

Leverage the law of large numbers: improve accuracy via repeated independent trials.

Hashing for Distinct Elements (Improved):

- Let $\mathbf{h} : U \to [0, 1]$ be a random hash function
- $\mathbf{s} := 1$
- For $i = 1, \ldots, n$
  - $\mathbf{s} := \min(\mathbf{s}, \mathbf{h}(x_i))$
- Return $\widehat{\mathbf{d}} = \frac{1}{\mathbf{s}} - 1$

Leverage the law of large numbers: improve accuracy via repeated independent trials.

Hashing for Distinct Elements (Improved):

- Let $\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_k : U \to [0, 1]$ be random hash functions
- $\mathbf{s} := 1$
- For $i = 1, \ldots, n$
  - $\mathbf{s} := \min(\mathbf{s}, \mathbf{h}(x_i))$
- Return $\widehat{\mathbf{d}} = \frac{1}{\mathbf{s}} - 1$

Leverage the law of large numbers: improve accuracy via repeated independent trials.

Hashing for Distinct Elements (Improved):

- Let $\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_k : U \to [0, 1]$ be random hash functions
- $\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_k := 1$
- For $i = 1, \ldots, n$
  - $\mathbf{s} := \min(\mathbf{s}, \mathbf{h}(x_i))$
- Return $\widehat{\mathbf{d}} = \frac{1}{\mathbf{s}} - 1$

Leverage the law of large numbers: improve accuracy via repeated independent trials.

Hashing for Distinct Elements (Improved):

- Let $\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_k : U \to [0, 1]$ be random hash functions
- $\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_k := 1$
- For $i = 1, \ldots, n$
  - For j=1,...,k, $\mathbf{s}_j := \min(\mathbf{s}_j, \mathbf{h}_j(x_i))$
- Return $\widehat{\mathbf{d}} = \frac{1}{\mathbf{s}} - 1$

Leverage the law of large numbers: improve accuracy via repeated independent trials.

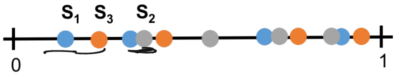
Hashing for Distinct Elements (Improved):

- Let $\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_k : U \to [0, 1]$ be random hash functions
- $\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_k := 1$
- For $i = 1, \ldots, n$
  - For j=1,...,k, $\mathbf{s}_j := \min(\mathbf{s}_j, \mathbf{h}_j(x_i))$
- $\mathbf{s} := \frac{1}{k} \sum_{j=1}^{k} \mathbf{s}_j$
- Return $\widehat{\mathbf{d}} = \frac{1}{\mathbf{s}} - 1$

Leverage the law of large numbers: improve accuracy via repeated independent trials.

Hashing for Distinct Elements (Improved):

- Let $\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_k : U \rightarrow [0, 1]$ be random hash functions
- $\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_k := 1$
- For $i = 1, \ldots, n$
  - For j=1,...,k, $\mathbf{s}_j := \min(\mathbf{s}_j, \mathbf{h}_j(x_i))$
- $\mathbf{s} := \frac{1}{k} \sum_{j=1}^{k} \mathbf{s}_j$
- Return $\widehat{\mathbf{d}} = \frac{1}{\mathbf{s}} - 1$



$\mathsf{s} = \frac{1}{k}\sum_{j=1}^{k} \mathsf{s}_j$. Have already shown that for $j = 1, \ldots, k$:

$$\mathbb{E}[\mathsf{s}_j] = \frac{1}{d+1}$$

$$\mathrm{Var}[\mathsf{s}_j] \leq \frac{1}{(d+1)^2}$$

s_j: minimum of d distinct hashes chosen randomly over $[0,1]$. $\mathsf{s} = \frac{1}{k}\sum_{j=1}^{k} \mathsf{s}_j$.
$\widehat{\mathsf{d}} = \frac{1}{\mathsf{s}} - 1$: estimate of # distinct elements d.

$s = \frac{1}{k}\sum_{j=1}^{k} s_j$. Have already shown that for $j = 1, \ldots, k$:

$$\mathbb{E}[s_j] = \frac{1}{d+1} \implies \mathbb{E}[s] = \frac{1}{k} \cdot \sum \mathbb{E} s_j = \frac{1}{k} \cdot k \cdot \frac{1}{d+1} = \frac{1}{d+1}$$

$$\mathrm{Var}[s_j] \leq \frac{1}{(d+1)^2}$$

s_j: minimum of d distinct hashes chosen randomly over $[0,1]$. $s = \frac{1}{k}\sum_{j=1}^{k} s_j$.
$\widehat{d} = \frac{1}{s} - 1$: estimate of # distinct elements d.

ANALYSIS

$\mathsf{s} = \frac{1}{k}\sum_{j=1}^{k} \mathsf{s}_j$. Have already shown that for $j = 1, \ldots, k$:

$$\mathbb{E}[\mathsf{s}_j] = \frac{1}{d+1} \implies \mathbb{E}[\mathsf{s}] = \frac{1}{d+1} \text{ (linearity of expectation)}$$

$$\mathrm{Var}[\mathsf{s}_j] \le \frac{1}{(d+1)^2}$$

s_j: minimum of d distinct hashes chosen randomly over $[0,1]$. $\mathsf{s} = \frac{1}{k}\sum_{j=1}^{k} \mathsf{s}_j$.
$\widehat{\mathsf{d}} = \frac{1}{\mathsf{s}} - 1$: estimate of # distinct elements d.

$\mathsf{s} = \frac{1}{k}\sum_{j=1}^{k} \mathsf{s}_j$. Have already shown that for $j = 1, \ldots, k$:

$$\mathbb{E}[\mathsf{s}_j] = \frac{1}{d+1} \implies \mathbb{E}[\mathsf{s}] = \frac{1}{d+1} \text{ (linearity of expectation)}$$

$$\text{Var}[\mathsf{s}_j] \leq \frac{1}{(d+1)^2} \implies \text{Var}[\mathsf{s}] = \frac{1}{k^2}\sum \text{Var}(s_j) = \frac{1}{k^2} \cdot k \cdot \frac{1}{(d+1)^2}$$

s_j: minimum of d distinct hashes chosen randomly over $[0, 1]$. $\mathsf{s} = \frac{1}{k}\sum_{j=1}^{k} \mathsf{s}_j$.
$\widehat{\mathsf{d}} = \frac{1}{\mathsf{s}} - 1$: estimate of # distinct elements d.

ANALYSIS

$\mathsf{s} = \frac{1}{k}\sum_{j=1}^{k} \mathsf{s}_j$. Have already shown that for $j = 1, \ldots, k$:

$$\mathbb{E}[\mathsf{s}_j] = \frac{1}{d+1} \implies \mathbb{E}[\mathsf{s}] = \frac{1}{d+1} \text{ (linearity of expectation)}$$

$$\mathrm{Var}[\mathsf{s}_j] \leq \frac{1}{(d+1)^2} \implies \mathrm{Var}[\mathsf{s}] \leq \frac{1}{k \cdot (d+1)^2} \text{ (linearity of variance)}$$

s_j: minimum of d distinct hashes chosen randomly over $[0,1]$. $\mathsf{s} = \frac{1}{k}\sum_{j=1}^{k} \mathsf{s}_j$.
$\widehat{\mathsf{d}} = \frac{1}{\mathsf{s}} - 1$: estimate of # distinct elements d.

$\mathsf{s} = \frac{1}{k}\sum_{j=1}^{k} \mathsf{s}_j$. Have already shown that for $j = 1, \ldots, k$:

$$\mathbb{E}[\mathsf{s}_j] = \frac{1}{d+1} \implies \mathbb{E}[\mathsf{s}] = \frac{1}{d+1} \text{ (linearity of expectation)}$$

$$\mathrm{Var}[\mathsf{s}_j] \leq \frac{1}{(d+1)^2} \implies \mathrm{Var}[\mathsf{s}] \leq \frac{1}{k \cdot (d+1)^2} \text{ (linearity of variance)}$$

Chebyshev Inequality:

$$\Pr\left[|\mathsf{s} - \mathbb{E}[\mathsf{s}]| \geq \epsilon \mathbb{E}[\mathsf{s}]\right] \leq \frac{\mathrm{Var}[\mathsf{s}]}{(\epsilon \mathbb{E}[\mathsf{s}])^2}$$

s_j: minimum of d distinct hashes chosen randomly over $[0, 1]$. $\mathsf{s} = \frac{1}{k}\sum_{j=1}^{k} \mathsf{s}_j$.
$\widehat{\mathsf{d}} = \frac{1}{\mathsf{s}} - 1$: estimate of # distinct elements d.

$\mathsf{s} = \frac{1}{k}\sum_{j=1}^{k}\mathsf{s}_j$. Have already shown that for $j = 1, \ldots, k$:

$$\mathbb{E}[\mathsf{s}_j] = \frac{1}{d+1} \implies \mathbb{E}[\mathsf{s}] = \frac{1}{d+1} \text{ (linearity of expectation)}$$

$$\mathrm{Var}[\mathsf{s}_j] \leq \frac{1}{(d+1)^2} \implies \mathrm{Var}[\mathsf{s}] \leq \frac{1}{k\cdot(d+1)^2} \text{ (linearity of variance)}$$

Chebyshev Inequality:

$$\Pr\left[|\mathsf{s} - \mathbb{E}[\mathsf{s}]| \geq \epsilon\mathbb{E}[\mathsf{s}]\right] \leq \frac{\mathrm{Var}[\mathsf{s}]}{(\epsilon\mathbb{E}[\mathsf{s}])^2} = \frac{\mathbb{E}[\mathsf{s}]^2/k}{\epsilon^2\mathbb{E}[\mathsf{s}]^2} = \frac{1}{k\cdot\epsilon^2}$$

s_j: minimum of d distinct hashes chosen randomly over $[0,1]$. $\mathsf{s} = \frac{1}{k}\sum_{j=1}^{k}\mathsf{s}_j$.
$\widehat{\mathsf{d}} = \frac{1}{\mathsf{s}} - 1$: estimate of # distinct elements d.

$\mathsf{s} = \frac{1}{k}\sum_{j=1}^{k}\mathsf{s}_j$. Have already shown that for $j = 1, \ldots, k$:

$$\mathbb{E}[\mathsf{s}_j] = \frac{1}{d+1} \implies \mathbb{E}[\mathsf{s}] = \frac{1}{d+1} \text{ (linearity of expectation)}$$

$$\mathrm{Var}[\mathsf{s}_j] \leq \frac{1}{(d+1)^2} \implies \mathrm{Var}[\mathsf{s}] \leq \frac{1}{k\cdot(d+1)^2} \text{ (linearity of variance)}$$

Chebyshev Inequality:

$$\Pr\left[\left|d - \widehat{\mathsf{d}}\right| \geq 4\epsilon\cdot d\right] \leq \frac{\mathrm{Var}[\mathsf{s}]}{(\epsilon\mathbb{E}[\mathsf{s}])^2} = \frac{\mathbb{E}[\mathsf{s}]^2/k}{\epsilon^2\mathbb{E}[\mathsf{s}]^2} = \frac{1}{k\cdot\epsilon^2}$$

s_j: minimum of d distinct hashes chosen randomly over $[0,1]$. $\mathsf{s} = \frac{1}{k}\sum_{j=1}^{k}\mathsf{s}_j$.
$\widehat{\mathsf{d}} = \frac{1}{\mathsf{s}} - 1$: estimate of # distinct elements d.

$\mathbf{s} = \frac{1}{k}\sum_{j=1}^{k} \mathbf{s}_j$. Have already shown that for $j = 1, \ldots, k$:

$$\mathbb{E}[\mathbf{s}_j] = \frac{1}{d+1} \implies \mathbb{E}[\mathbf{s}] = \frac{1}{d+1} \text{ (linearity of expectation)}$$

$$\text{Var}[\mathbf{s}_j] \le \frac{1}{(d+1)^2} \implies \text{Var}[\mathbf{s}] \le \frac{1}{k \cdot (d+1)^2} \text{ (linearity of variance)}$$

Chebyshev Inequality:

$$\Pr\left[\left|d - \widehat{\mathbf{d}}\right| \ge 4\epsilon \cdot d\right] \le \frac{\text{Var}[\mathbf{s}]}{(\epsilon\mathbb{E}[\mathbf{s}])^2} = \frac{\mathbb{E}[\mathbf{s}]^2/k}{\epsilon^2\mathbb{E}[\mathbf{s}]^2} = \frac{1}{k \cdot \epsilon^2} = \delta$$

How should we set k if we want $4\epsilon \cdot d$ error with probability $\ge 1 - \delta$?

$$\frac{1}{k\epsilon^2} = \delta \qquad k = \frac{1}{\epsilon^2 \delta}$$

$\mathbf{s}_j$: minimum of d distinct hashes chosen randomly over $[0, 1]$. $\mathbf{s} = \frac{1}{k}\sum_{j=1}^{k} \mathbf{s}_j$.
$\widehat{\mathbf{d}} = \frac{1}{\mathbf{s}} - 1$: estimate of # distinct elements d.

$\mathbf{s} = \frac{1}{k}\sum_{j=1}^{k} \mathbf{s}_j$. Have already shown that for $j = 1, \ldots, k$:

$$\mathbb{E}[\mathbf{s}_j] = \frac{1}{d+1} \implies \mathbb{E}[\mathbf{s}] = \frac{1}{d+1} \text{ (linearity of expectation)}$$

$$\mathrm{Var}[\mathbf{s}_j] \leq \frac{1}{(d+1)^2} \implies \mathrm{Var}[\mathbf{s}] \leq \frac{1}{k \cdot (d+1)^2} \text{ (linearity of variance)}$$

Chebyshev Inequality:

$$\Pr\left[\left|d - \widehat{\mathbf{d}}\right| \geq 4\epsilon \cdot d\right] \leq \frac{\mathrm{Var}[\mathbf{s}]}{(\epsilon \mathbb{E}[\mathbf{s}])^2} = \frac{\mathbb{E}[\mathbf{s}]^2/k}{\epsilon^2 \mathbb{E}[\mathbf{s}]^2} = \frac{1}{k \cdot \epsilon^2}$$

How should we set k if we want $4\epsilon \cdot d$ error with probability $\geq 1 - \delta$?

$k = \frac{1}{\epsilon^2 \cdot \delta}$.

$\mathbf{s}_j$: minimum of d distinct hashes chosen randomly over $[0, 1]$. $\mathbf{s} = \frac{1}{k}\sum_{j=1}^{k} \mathbf{s}_j$.
$\widehat{\mathbf{d}} = \frac{1}{\mathbf{s}} - 1$: estimate of # distinct elements d.

$\mathsf{s} = \frac{1}{k}\sum_{j=1}^{k} \mathsf{s}_j$. Have already shown that for $j = 1, \ldots, k$:

$$\mathbb{E}[\mathsf{s}_j] = \frac{1}{d+1} \implies \mathbb{E}[\mathsf{s}] = \frac{1}{d+1} \text{ (linearity of expectation)}$$

$$\mathrm{Var}[\mathsf{s}_j] \le \frac{1}{(d+1)^2} \implies \mathrm{Var}[\mathsf{s}] \le \frac{1}{k \cdot (d+1)^2} \text{ (linearity of variance)}$$

Chebyshev Inequality:

$$\Pr\left[\left|d - \widehat{\mathsf{d}}\right| \ge 4\epsilon \cdot d\right] \le \frac{\mathrm{Var}[\mathsf{s}]}{(\epsilon\mathbb{E}[\mathsf{s}])^2} = \frac{\mathbb{E}[\mathsf{s}]^2/k}{\epsilon^2\mathbb{E}[\mathsf{s}]^2} = \frac{1}{k \cdot \epsilon^2} = \frac{\epsilon^2 \cdot \delta}{\epsilon^2} = \delta.$$

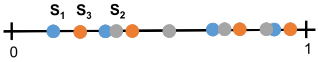
How should we set k if we want $4\epsilon \cdot d$ error with probability $\ge 1 - \delta$?

$k = \frac{1}{\epsilon^2 \cdot \delta}$.

s_j: minimum of d distinct hashes chosen randomly over $[0, 1]$. $\mathsf{s} = \frac{1}{k}\sum_{j=1}^{k} \mathsf{s}_j$.
$\widehat{\mathsf{d}} = \frac{1}{\mathsf{s}} - 1$: estimate of # distinct elements d.

Hashing for Distinct Elements:

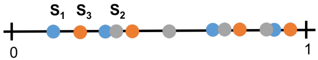
- Let $\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_k : U \to [0, 1]$ be random hash functions
- $\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_k := 1$
- For $i = 1, \ldots, n$
  - For j=1,..., k, $\mathbf{s}_j := \min(\mathbf{s}_j, \mathbf{h}_j(x_i))$
- $\mathbf{s} := \frac{1}{k}\sum_{j=1}^{k} \mathbf{s}_j$
- Return $\widehat{\mathbf{d}} = \frac{1}{\mathbf{s}} - 1$



- Setting $k = \frac{1}{\epsilon^2 \cdot \delta}$, algorithm returns $\widehat{\mathbf{d}}$ with $|d - \widehat{\mathbf{d}}| \leq 4\epsilon \cdot d$ with probability at least $1 - \delta$.

Hashing for Distinct Elements:

- Let $\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_k : U \to [0, 1]$ be random hash functions
- $\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_k := 1$
- For $i = 1, \ldots, n$
  - For j=1,..., k, $\mathbf{s}_j := \min(\mathbf{s}_j, \mathbf{h}_j(x_i))$
- $\mathbf{s} := \frac{1}{k} \sum_{j=1}^{k} \mathbf{s}_j$
- Return $\widehat{\mathbf{d}} = \frac{1}{\mathbf{s}} - 1$

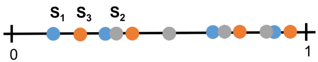


- Setting $k = \frac{1}{\epsilon^2 \cdot \delta}$, algorithm returns $\widehat{\mathbf{d}}$ with $|d - \widehat{\mathbf{d}}| \leq 4\epsilon \cdot d$ with probability at least $1 - \delta$.
- Space complexity is $k = \frac{1}{\epsilon^2 \cdot \delta}$ real numbers $s_1, \ldots, s_k$.

SPACE COMPLEXITY

Hashing for Distinct Elements:

- Let $\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_k : U \to [0, 1]$ be random hash functions
- $\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_k := 1$
- For $i = 1, \ldots, n$
  - For j=1,..., k, $\mathbf{s}_j := \min(\mathbf{s}_j, \mathbf{h}_j(x_i))$
- $\mathbf{s} := \frac{1}{k} \sum_{j=1}^{k} \mathbf{s}_j$
- Return $\widehat{\mathbf{d}} = \frac{1}{\mathbf{s}} - 1$



x = y
h(x) = h(y)
w.p. 1

x ≠ y
h(x) = h(y)
with small prob.

- Setting $k = \frac{1}{\epsilon^2 \cdot \delta}$, algorithm returns $\widehat{\mathbf{d}}$ with $|d - \widehat{\mathbf{d}}| \leq 4\epsilon \cdot d$ with probability at least $1 - \delta$.

$\frac{1}{\delta} = 20$

- Space complexity is $k = \frac{1}{\epsilon^2 \cdot \delta}$ real numbers $s_1, \ldots, s_k$.
- $\delta = 5\%$ failure rate gives a factor 20 overhead in space complexity.