CS-GY 6763: Lecture 2
Hashing + Fingerprinting, Chebyshev's Inequality

NYU Tandon School of Engineering, Prof. Christopher Musco

NOTE ON MATHEMATICAL PROOFS

It can be hard to know how formal to be. We will try to provide feedback on first problem set for anyone who is either too rigorous or too loose. It's a learning process.

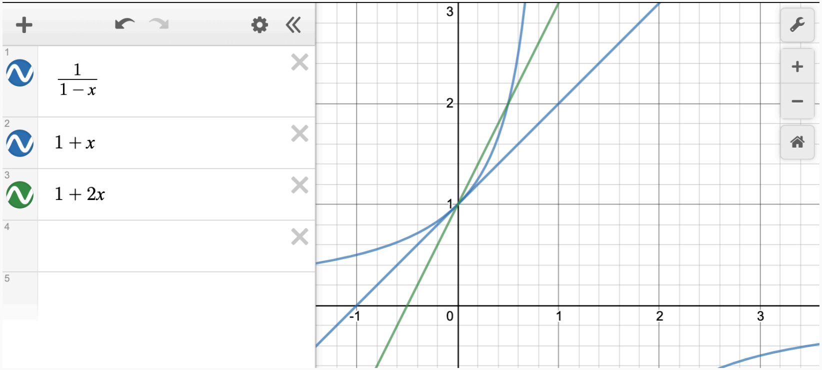
Things that are generally fine:

- Can assume input size n is $> C$ for some constant c. E.g. $n > 2, n > 10$.
- Similarly can assume $\epsilon < c$ for constant c. E.g. $\epsilon < .1, \epsilon < .01$.
- If I write $O(z)$, you are free to choose the constant. E.g., it's fine if your analysis of CountSketch only works for tables of size $1000 \cdot m$.
- Derivatives, integrals, etc. can be taken from e.g. WolframAlpha without working through steps.
- Basic inequalities can be used without proof, as long as you verify numerically. Don't need to include plot on problem set.

EXAMPLE INEQUALITY

$$1 + \epsilon \leq \frac{1}{1-\epsilon} \leq 1 + 2\epsilon \text{ for } \epsilon \in [0, .5].$$

Proof by plotting:


$\frac{1}{1-x}$

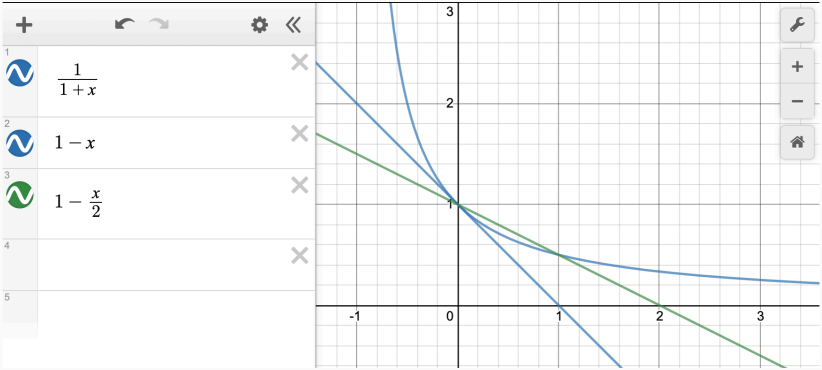
$1 + x$

$1 + 2x$

EXAMPLE INEQUALITY

$$1 - \epsilon \leq \frac{1}{1+\epsilon} \leq 1 - .5\epsilon \text{ for } \epsilon \in [0,1].$$

Proof by plotting:



1 $\frac{1}{1+x}$

2 $1-x$

3 $1-\frac{x}{2}$

4

5

GENERAL ADVICE

Tip: When confronted with a complex expression, try to simplify by using big-Oh notation, or just rounding things off. Then clean-up your proof after you get to a solution.

Examples:

- To start: $(m-1) \approx m$. Later: $m/2 \leq m-1 \leq m$.
- To start: $\frac{1}{n} - \frac{1}{n^2} \approx \frac{1}{n}$. Later: $\frac{1}{2n} \leq \frac{1}{n} - \frac{1}{n^2} \leq \frac{1}{n}$.
- $\log(n/2) \approx \log(n)$ Later: $\log(n)/2 \leq \log(n/2) \leq \log(n)$.

DEFINITIONS OF INDEPENDENCE

Suppose we have random variables $X_1, \ldots, X_k$. We say that **a pair of random variables** X_i and X_j are <u>independent</u> if, for all possible values v_i, v_j,

$$\Pr[X_i = v_i \text{ and } X_j = v_j] = \Pr[X_i = v_i] \cdot \Pr[X_j = v_j].$$

We say $X_1, \ldots, X_k$ are <u>pairwise independent</u> if X_i, X_j are independent for all $i, j \in \{1, \ldots, k\}$.

We say $X_1, \ldots, X_k$ are <u>mutually independent</u> if, for all possible values $v_1, \ldots, v_k$,

$$\Pr[X_1 = v_1, \ldots, X_k = v_k] = \Pr[X_1 = v_1] \cdot \ldots \cdot \Pr[X_k = v_k].$$

Mutual independence implies pairwise independence, but pairwise independence does not imply mutual independence.

DEFINITIONS OF INDEPENDENCE

Give an example of three random variables that are pairwise independent but not mutually independent.

$X_1, \ldots, X_k$ are pairwise independent if for all i, j, v_i, v_j,

$$\Pr[X_i = v_i \text{ and } X_j = v_j = \Pr[X_i = v_i] \cdot \Pr[X_j = v_j].$$

$X_1, \ldots, X_k$ are mutually independent if, for all $v_1, \ldots, v_k$,

$$\Pr[X_1 = v_1, \ldots, X_k = v_k] = \Pr[X_1 = v_1] \cdot \ldots \cdot \Pr[X_k = v_k].$$

LINEARITY OF VARIANCE

If we have two independent random variables X, Y, then:

$$\mathrm{Var}[X+Y] = \mathrm{Var}[X] + \mathrm{Var}[Y].$$

If we have a set of pairwise independent[1] random variables $X_1, \ldots, X_k$ then:

$$\mathrm{Var}\left[\sum_{i=1}^{k} X_i\right] = \sum_{i=1}^{k} \mathrm{Var}[X_i].$$

Mutual independence is not necessary!

[1]Technically, pairwise uncorrelated suffices, which is a weaker assumption.

UNIFORMLY RANDOM HASH FUNCTION

Let h be a random function from $|\mathcal{U}| \to \{1, \ldots, m\}$. This means that h is constructed by an algorithm using a seed of random numbers, but then the function is fixed. Given input $x \in \mathcal{U}$, it always returns the same output, $h(x)$.

Recall: Uniformly Random Hash Function. A random function $h : \mathcal{U} \to \{1, \ldots, m\}$ is called uniformly random if:

- $\Pr[h(x) = i] = \frac{1}{m}$ for all $x \in \mathcal{U}$, $i \in \{1, \ldots, m\}$.
- $h(x), h(y), h(z), \ldots$ are mutually independent random variables for all $x, y, z, \ldots \in \mathcal{U}$.
    - Which implies that $\Pr[h(x) = h(y)] =$
    - Which implies that $\Pr[h(x) = h(y) = h(z)] =$

UNIFORMLY RANDOM HASH FUNCTION

The only way to implement a truly random hash function is to create a giant lookup table, where the numbers on the right are chosen independently at random from $\{1, \ldots, m\}$.

x	h(x)
1	14
2	25
3	99
4	16
$\vdots$	$\vdots$
$\lvert\mathcal{U}\rvert$	87

If we're hashing 35 char ASCII strings (e.g. urls) the length of the table is greater than the number of atoms in the universe.

UNIVERSAL HASH FUNCTIONS

For the application to CountMin from last class we can weaken our assumption that h is uniformly random.

Definition (Universal hash function)

A random hash function $h : \mathcal{U} \to \{1, \ldots, m\}$ is universal if, for any fixed $x, y \in \mathcal{U}$,

$$\Pr[h(x) = h(y)] \leq \frac{1}{m}.$$

Claim: A uniformly random hash-function is universal.

UNIVERSAL HASH FUNCTIONS

Definition (Universal hash function)

A random hash function $h : \mathcal{U} \to \{1, \ldots, m\}$ is universal if, for any fixed $x, y \in \mathcal{U}$,

$$\Pr[h(x) = h(y)] \leq \frac{1}{m}.$$

Efficient alternative: Let p be a prime number between $|\mathcal{U}|$ and $2|\mathcal{U}|$. Let a, b be random numbers in $0, \ldots, p$, $a \neq 0$.

$$h(x) = [a \cdot x + b \pmod p] \pmod m$$

is universal. Lecture notes with proof posted on website. Requires some abstract algebra.

How much space does this hash function take to store?

LIMITED INDEPENDENCE HASH FUNCTIONS

Similar alternative definition:

Definition (Pairwise independent hash function)

A random hash function $h : \mathcal{U} \to \{1, \ldots, m\}$ is pairwise independent if, for any fixed $x, y \in \mathcal{U}, i, j \in \{1 \ldots, m\}$,

$$\Pr[h(x) = i \text{ and } h(y) = j] = \frac{1}{m^2}.$$

Basically same construction as universal hash, except we don't restrict $a \neq 0$ and m to be a prime power.

Claim: A pairwise independent hash-function is universal.

LIMITED INDEPENDENCE HASH FUNCTIONS

Definition (k-wise independent hash function)

A random hash function $h : \mathcal{U} \to \{1, \ldots, m\}$ is k-wise independent if, for any fixed $x, y \in \mathcal{U}, i, j \in \{1 \ldots, m\}$,

$$\Pr[h(u_1) = v_1 \text{ and } h(u_2) = v_2 \text{ and } \ldots h(u_k) = v_k] = \frac{1}{m^k},$$

for all $u_1, \ldots, u_k \in \mathcal{U}$ and $v_1, \ldots, v_k \in \{1, \ldots, m\}$.

Strictly stronger than pairwise independence and needed for some applications. But we will never need $k > O(\log n)$ in this class.

Example: For random coefficients $c_0, \ldots, c_k \in \{0, \ldots, p\}$,

$$h(x) = \left[c_0 + c_1 x + c_2 x^2 + \ldots c_k x^k \pmod{p}\right] \pmod{m}$$

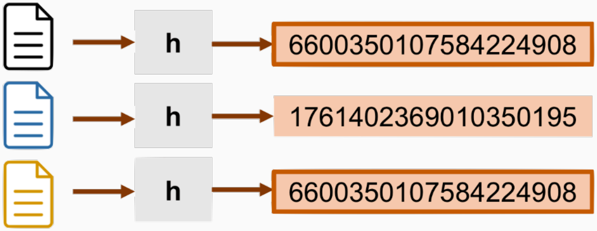
PSEUDORANDOM HASH FUNCTIONS

We won’t prove that random polynomials provide good hash functions, but I want to give a flavor of what is involved (e.g., why do prime numbers show up?).

FINGERPRINTING

Goal: Construct a compact "fingerprint" $h(f)$ for any given file f with two properties:

- The fingerprints $h(f_1)$ and $h(f_2)$ should be different with high probability if the contents of f_1 and f_2 differ at all.
- If the contents of f_1 and f_2 are identical, we should have $h(f_1) = h(f_2)$.

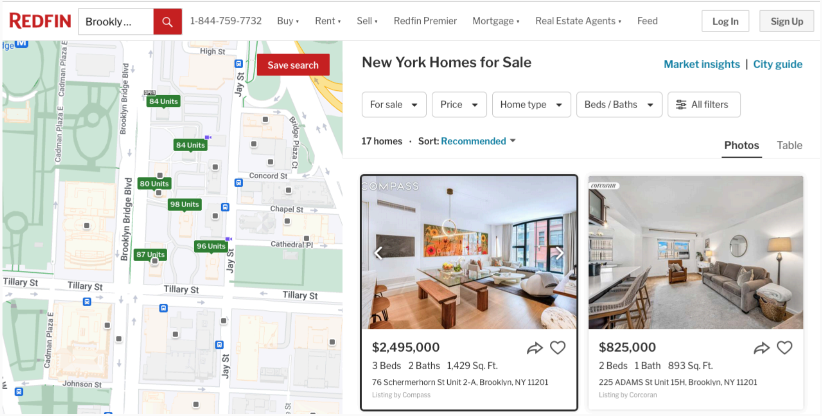


(Basically the same goal as most applications of hashing.)

APPLICATIONS OF FINGER PRINTING

- Quickly check if two versions of the same file are identical (e.g. in version control systems like Git). Do not need to communicate the entire file between servers. Also used in webcaching and content delivery networks.
- Check that a file pieced together from multiple parts is not missing anything.

APPLICATIONS OF FINGER PRINTING



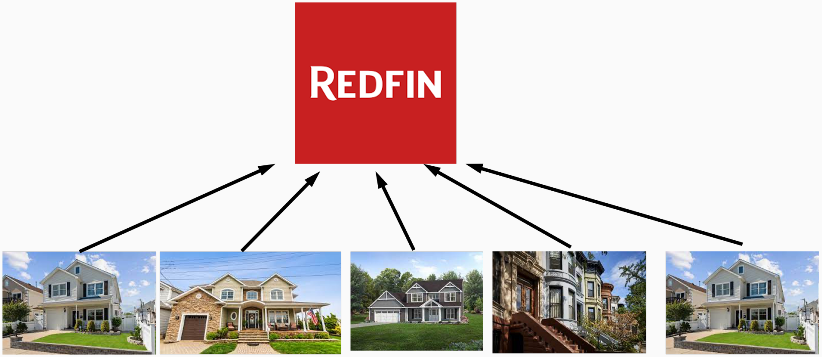
New York Homes for Sale

17 homes · Sort: Recommended

$2,495,000
3 Beds 2 Baths 1,429 Sq. Ft.
76 Schermerhorn St Unit 2-A, Brooklyn, NY 11201
Listing by Compass

$825,000
2 Beds 1 Bath 893 Sq. Ft.
225 ADAMS St Unit 15H, Brooklyn, NY 11201
Listing by Corcoran

APPLICATIONS OF FINGER PRINTING



Images from databases of local real estate agencies.

Fingerprints used as file names for the images to make sure we did not reupload new images that we already had, and to detect duplicate images and listings.

FINGERPRINTING

Goal: Construct a compact “fingerprint” function $h(f)$ such that:

- $h(f_1) \neq h(f_2)$ if $f_1 \neq f_2$ with high probability.

Ideally, length of $h(f_1)$ (i.e. the size of the integers hashed to) is much less than the file size.

RANDOM FINGERPRINTING

Rabin Fingerprint (1981): Let file $f = 010\ldots1101$ of length n be interpreted as an n bit integer. So something between 0 and 2^n.

Construct h randomly: Choose random prime number p between 2 and $tn\log(tn)$ for a constant t.

$$h(f) = f \pmod{p}.$$

How many bits does $h(f)$ take to store?

RANDOM FINGERPRINTING

$$h(f) = f \pmod p \quad \text{for random prime } p \in \{2, \ldots, tn\log(tn)\}$$

Claim: If $f_1 \neq f_2$ then $h(f_1) = h(f_2)$ with probability $\leq \frac{2}{t}$.

Since our fingerprint only takes $O(\log n + \log t)$ space, we can set t to be super large, so effectively the probability of $h(f_1)$ and $h(f_2)$ colliding is negligible for all real-world applications.

E.g. set fingerprint length to $\log n + 28$ bits and you are more likely to win Megamillions.

RANDOM FINGERPRINTING

How do we sample a random prime between $2, \ldots, tn \log n$?

Keep in mind that n is pretty large here. For a 200kb image, $n \approx 1.6$ million.

FROM PRIME TESTING TO PRIME GENERATION

Rejection sampling:

- Pick a random q bit number.
- Check if it's prime. Can be done in $O(q^3)$ time.
- If not, repeat.

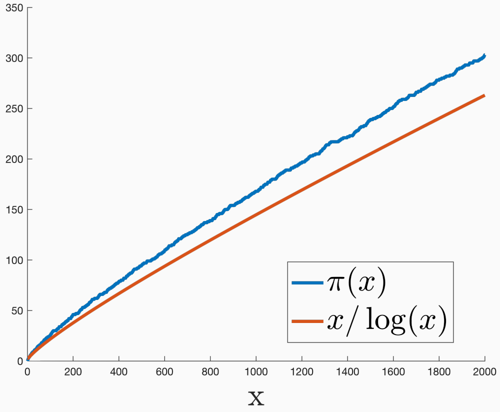
Here we would have $q \approx \log(tn \log n) \approx 48$ for the example above. So each iteration is efficient, but is this efficient overall?

Roughly how many tries do you expect this to take?

PRIME NUMBER THEOREM

Let $\pi(x)$ denote the number of primes less than some integer x. **Informally:**

$$\pi(x) \sim \frac{x}{\ln(x)}$$



PRIME NUMBER THEOREM

Formally: For $x > 17$,

$$\frac{x}{\ln(x)} \leq \pi(x) \leq \frac{x}{\ln(x) - 4}$$

So if we select a random $q = 48$ bit number, the chance that it is prime is great than:

$$\frac{1}{\ln(2^q)} \geq \frac{1}{34}$$

After a few hundred tries, we will almost definitely find a prime number. **In general, need $O(q)$ tries in expectation to find a prime with q bits.**

Remark: Finding large prime numbers is important in some other applications beyond hashing.

RANDOM FINGERPRINTING

$$h(f) = f \pmod{p} \quad \text{for prime } p \in \{2, \ldots, tn\log(tn)\}$$

Claim: If $f_1 \neq f_2$ then $h(f_1) = h(f_2)$ with probability $\leq \frac{2}{t}$.

First observation: If $h(f_1) = h(f_2)$, then:

$$(f_1 - f_2) \pmod{p} = 0.$$

In other words, we only fail $\underline{\text{if } |f_1 - f_2| \text{ is divisible by } p}$.

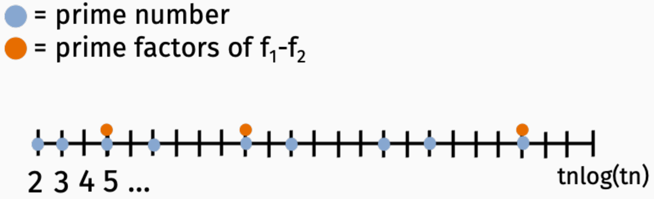
RANDOM FINGERPRINTING

Question: What is the chance that $|f_1 - f_2|$ is divisible by a random prime $p \in \{2, \ldots, tn\log(tn)\}$?

RANDOM FINGERPRINTING

Number of distinct prime factors of $f_1 - f_2$: At most n.

Number of primes between $\{2, \ldots, tn\log(tn)\}$: At least $\frac{tn\log(tn)}{\log(tn\log(tn))}$ via prime number theorem.



Chance we pick a prime factor of $f_1 - f_2$ is less than:

$$\frac{n}{\frac{tn\log(tn)}{\log(tn\log(tn))}} = \frac{\log(tn\log(tn))}{t\log(tn)} \leq \frac{2\log(tn)}{t\log(tn)}$$

RANDOM FINGERPRINTING

Conclusion: The chance that a random prime $p \in \{2, \ldots, tn\log(tn)\}$ is a factor of $|f_1 - f_2|$ is $\leq \frac{2}{t}$.

So, for two files $f_1 \neq f_2$, the chance that $h(f_1) = h(f_2) \leq \frac{2}{t}$.

Set $t = 2^{28}$ (the chance you win Megamillions).

Fingerprint size: At most $2\log_2(nt) = 2\log_2(n) + 2\log_2(2^{28})$ bits.

Suppose we are fingerprinting 200kb image files. $n \approx 1,600,000$, so our fingerprint has size:

96 bits

This amounts to a 17,000x reduction over sending and comparing the original files.

REMAINDER OF LECTURE

Last week we saw the power of Linearity of Expectation + Markov's. This week we will discuss two more tools:

- Linearity of Variance + Chebyshev's Inequality

Next week:

- Union Bound + Exponential Tail Bounds

These six tools combined are surprising powerful and flexible. They form the cornerstone of randomized algorithm design.

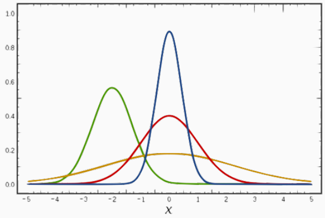
CHEBYSHEV'S INEQUALITY

A new concentration inequality:

Lemma (Chebyshev's Inequality)

Let X be a random variable with expectation $\mathbb{E}[X]$ and variance $\sigma^2 = \text{Var}[X]$. Then for any $k > 0$,

$$\Pr[|X - \mathbb{E}[X]| \geq k \cdot \sigma] \leq \frac{1}{k^2}$$



$\sigma = \sqrt{\text{Var}[X]}$ is the standard deviation of X. Intuitively this bound makes sense: it is tighter when σ is smaller.

COMPARISON TO MARKOV'S INEQUALITY

Properties of Chebyshev's inequality:

- **Good:** No requirement of non-negativity. X can be anything.
- **Good:** Two-sided. Bounds the probability that $|X - \mathbb{E}X|$ is large, which means that X isn't too far above or below its expectation. Markov's only bounded probability that X exceeds $\mathbb{E}[X]$.
- **Bad/Good:** Requires a bound on the variance of of X.

No hard rule for which to apply! Both Markov's and Chebyshev's are useful in different settings.

PROOF OF CHEBYSHEV'S INEQUALITY

Idea: Apply Markov's inequality to the (non-negative) random variable $S = (X - \mathbb{E}[X])^2$.

Lemma (Chebyshev's Inequality)

Let X be a random variable with expectation $\mathbb{E}[X]$ and variance $\sigma^2 = \mathrm{Var}[X]$. Then for any $k > 0$,

$$\Pr[|X - \mathbb{E}[X]| \geq k \cdot \sigma] \leq \frac{1}{k^2}$$

Markov's inequality: for non-negative r.v. S, $\Pr[S \geq t] \leq \mathbb{E}[S]/t$.

QUICK EXAMPLE

If I flip a fair coin 100 times, show that with $> 93\%$ chance I get between 30 and 70 heads.

Let $C_1, \ldots, C_{100}$ be independent random variables that are 1 with probability 1/2, 0 otherwise.

Let $H = \sum_{i=1}^{100} C_i$ be the number of heads that get flipped.

$$\mathbb{E}[H] =$$

$$\text{Var}[H] =$$

QUICK EXAMPLE

If I flip a fair coin 100 times, show that with 93% chance I get between 30 and 70 heads?

Let $C_1, \ldots, C_{100}$ be independent random variables that are 1 with probability 1/2, 0 otherwise.

Let $H = \sum_{i=1}^{100} C_i$ be the number of heads that get flipped.

$\mathbb{E}[H] = 50$, $\mathrm{Var}[H] = 25$.

Chebyshev's:

APPLICATIONS OF CHEBYSHEVS INEQUALITY

Abstract architecture of a streaming algorithm:

Have massive dataset $X = x_1, \ldots, x_n$ with n pieces of data that arrive in a sequential stream. There is far too much data to store or process it in a single location.

- Still want to analyze the data: i.e. fit a model or (approximately) compute some function $f(X)$.
- To do so, we must compress data "on-the-fly", storing some smaller data structure which still contains interesting information.
- Often can only take a <u>single-pass</u> over the data.

Count-Min was our first example of a streaming algorithm for the (ϵ, k)-frequent items problem.

STREAMING ALGORITHMS IN PRACTICE

Sensor data: GPS or seismometer readings to detect geological anomalies, telescope images, satellite imagery, highway travel time sensors.

Web traffic and data: User data for website, including e.g. click data, web searches and API queries, posts and image uploads on social media.

Training machine learning models: Often done in a streaming setting when training dataset is huge, often with multiple passes.

STORM Flink APACHE kafka samza

Lots of software frameworks exist for easy development of streaming algorithms.

distinct elements problem

Input: $x_1, \ldots, x_n \in \mathcal{U}$ where $\mathcal{U}$ is a huge universe of items.

Output: Number of distinct inputs.

Example: $f(1, 10, 2, 4, 9, 2, 10, 4) \rightarrow 5$

Applications:

- Distinct users hitting a webpage.
- Distinct users using a new feature or UI in a certain way.
- Distinct values in a database column (e.g. for estimating the size of group by queries)
- Number of distinct queries to a search engine.
- Distinct motifs in DNA sequence.

Implementations widely used at Google (Sawzall, Dremel, PowerDrill), Twitter, Facebook (Presto), etc.

DISTINCT ELEMENTS PROBLEM

Input: $d_1, \ldots, d_n \in \mathcal{U}$ where $\mathcal{U}$ is a huge universe of items.

Output: Number of distinct inputs, D.

Example: $f(1, 10, 2, 4, 9, 2, 10, 4) \rightarrow D = 5$

Naive Approach: Store a dictionary of all items seen so far. Takes $O(D)$ space. We will aim to do a lot better than that.

Goal: Return $\tilde{D}$ satisfying

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

using only $O(1/\epsilon^2)$ space.

distinct elements problem

Input: $d_1, \ldots, d_n \in \mathcal{U}$ where $\mathcal{U}$ is a huge universe of items.

Output: Number of distinct inputs, D.

Example: $f(1, 10, 2, 4, 9, 2, 10, 4) \rightarrow D = 5$

Flajolet–Martin (simplified):

```
• Choose random hash function $h : \mathcal{U} \rightarrow [0, 1]$.
• $S = 1$
• For $i = 1, \ldots, n$
    • $S \leftarrow \min(S, h(x_i))$
• Return: $\frac{1}{S} - 1$
```

HOLD UP...

The hash function h maps from $\mathcal{U}$ to a random point in $[0, 1]$?

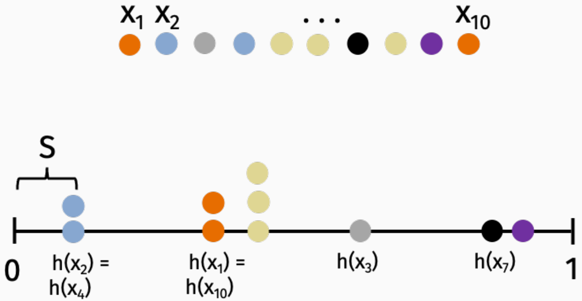
Hashing to real numbers:

- Impossible to implement $h(x)$ in reality, but you can replace it with $\frac{g(x)}{k}$, where g is a hash function that maps to $\{0, 1, \ldots, k\}$ for sufficiently large k.
- All results hold if this “discrete” hash is used instead, but the analysis is simpler if we assume access to h.
- Just like when we assumed uniform random hash functions, this is a useful abstraction which makes understanding and analyzing algorithms easier.

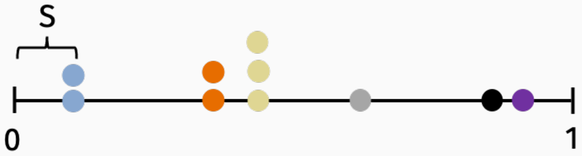
VISUALIZATION

Flajolet–Martin (simplified):

- Choose random hash function $h : \mathcal{U} \to [0, 1]$.
- $S = 1$
- For $i = 1, \ldots, n$
    - $S \leftarrow \min(S, h(x_i))$
- Return: $\tilde{D} = \frac{1}{S} - 1$



VISUALIZATION

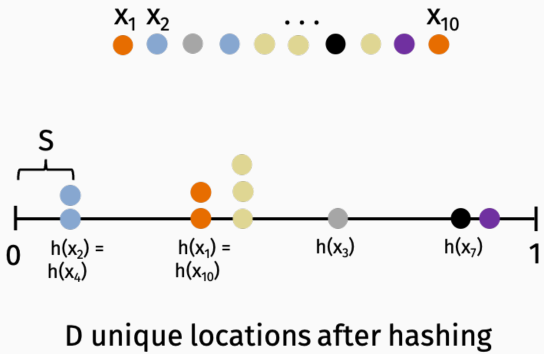


Important: If $\mathcal{U} \rightarrow [0, 1]$ uniformly at random, we can assume that there are no collisions. If we instead used a discrete grid, it would suffice to use a hash table of size $O(D^2)$ or, conservatively, $O(|\mathcal{U}|^2)$.

We will not do a formal analysis, but roughly how many bits does S takes to store?

FM ANALYSIS

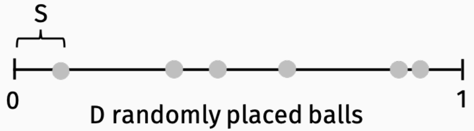
Let D equal the number of distinct elements in our stream.



Intuition: When D is larger, S will be smaller. Makes sense to return the estimate $\tilde{D} = \frac{1}{S} - 1$.

FM ANALYSIS

What is $\mathbb{E}S$?



Let D equal the number of distinct elements in our stream.

Lemma

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

THE CALCULUS PROOF

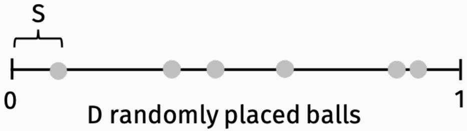
Proof:

$$
\begin{aligned}
\mathbb{E}[S] &= \int_0^1 \Pr[S \geq \lambda] d\lambda \\
&= \int_0^1 (1-\lambda)^D d\lambda \\
&= \frac{-(1-\lambda)^{D+1}}{D+1} \Big|_{\lambda=0}^{1} \\
&= \frac{1}{D+1}
\end{aligned}
$$

Exercise: Why?

PROOF "FROM THE BOOK"

$\mathbb{E}[S] = \Pr[(D+1)^{\text{st}}$ item has the smallest hash value].

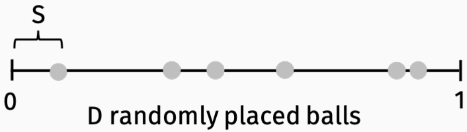


Formally, we are using the fact that:

$$\Pr[A] = \mathbb{E}_{h_1,\ldots,h_D}\left[\Pr\left[A \mid h_1,\ldots,h_D\right]\right]$$

PROOF "FROM THE BOOK"

$\mathbb{E}[S] = \Pr[(D+1)^{\text{st}}$ item has the smallest hash value$]$.



By symmetry, this equals $\frac{1}{D+1}$ (since every ball is equally likely to be first).

PROVING CONCENTRATION

$\mathbb{E}S = \frac{1}{D+1}$. **Estimate:** $\tilde{D} = \frac{1}{S} - 1$. We have for $\epsilon < \frac{1}{2}$:

$$\text{If } (1-\epsilon)\mathbb{E}S \leq S \leq (1+\epsilon)\mathbb{E}S, \text{ then:}$$

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

So, it suffices to show that S concentrates around its mean. I.e. that $|S - \mathbb{E}S| \leq \epsilon \cdot \mathbb{E}S$. We will use Chebyshev's inequality as our concentration bound.

ϵ MANIPULATION TRICKS

Recall:

$$1+\epsilon \leq \frac{1}{1-\epsilon} \leq 1+2\epsilon \text{ for } \epsilon \in [0, .5].$$

$$1-\epsilon \leq \frac{1}{1+\epsilon} \leq 1-.5\epsilon \text{ for } \epsilon \in [0, 1].$$

CALCULUS PROOF

Lemma

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

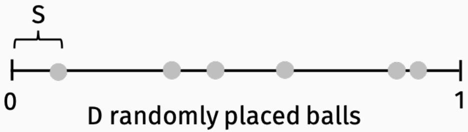
Proof:

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

`www.wolframalpha.com/input?i=antiderivative+of+%281-sqrt%28x%29%29%5ED`

PROOF "FROM THE BOOK"

$\mathbb{E}[S^2] = ??.$



FM ANALYSIS

Recall we want to show that, with high probability, $(1-\epsilon)\mathbb{E}[S] \leq S \leq (1-\epsilon)\mathbb{E}[S]$.

- $\mathbb{E}[S] = \frac{1}{D+1} = \mu$.
- $\mathrm{Var}[S] \leq \frac{1}{(D+1)^2} = \mu^2$. Standard deviation: $\sigma \leq \mu$.
- Want to bound $\Pr[|S-\mu| \geq \epsilon\mu] \leq \delta$.

Chebyshev's: $\Pr[|S-\mu| \geq \epsilon\mu] = \Pr[|S-\mu| \geq \epsilon\sigma] \leq \frac{1}{\epsilon^2}$.

Vacuous bound. Our variance is way too high!

VARIANCE REDUCTION

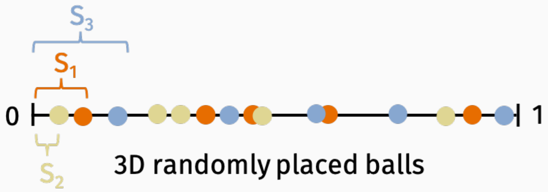
Trick of the trade: Repeat many independent trials and take the mean to get a better estimator.

Given i.i.d. (independent, identically distributed) random variables $X_1, \ldots, X_k$ with mean μ and variance σ^2, what is:

- $\mathbb{E}\left[\frac{1}{k}\sum_{i=1}^{k} X_i\right] =$
- $\text{Var}\left[\frac{1}{k}\sum_{i=1}^{k} X_i\right] =$

FM ANALYSIS

Using independent hash functions, maintain k independent sketches $S_1, \ldots, S_k$.



Flajolet–Martin:

- Choose k random hash function $h_1, \ldots, h_k : \mathcal{U} \rightarrow [0,1]$.
- $S_1 = 1, \ldots, S_k = 1$
- For $i = 1, \ldots, n$
    - $S_j \leftarrow \min(S_j, h_j(x_i))$ for all $j \in 1, \ldots, k$.
- $S = (S_1 + \ldots + S_k)/k$
- Return: $\frac{1}{S} - 1$

FM ANALYSIS

1 estimator:

- $\mathbb{E}[S] = \frac{1}{D+1} = \mu$.
- $\mathrm{Var}[S] = \mu^2$

k estimators:

- $\mathbb{E}[S] = \frac{1}{D+1} = \mu$.
- $\mathrm{Var}[S] \leq \mu^2/k$
- By Chebyshev, $\Pr[|S - \mathbb{E}S| \geq c\mu/\sqrt{k}] \leq \frac{1}{c^2}$.

Setting $c = 1/\sqrt{\delta}$ and $k = \frac{1}{\epsilon^2\delta}$ gives:

$$\Pr[|S - \mu| \geq \epsilon\mu] \leq \delta.$$

Total space complexity: $O\left(\frac{1}{\epsilon^2\delta}\right)$ to estimate distinct elements up to error ϵ with success probability $1 - \delta$.

FM ANALYSIS

Total space complexity: $O\left(\frac{1}{\epsilon^2\delta}\right)$ to estimate distinct elements up to error ϵ with success probability $1-\delta$.

- Recall that to ensure $(1-\bar{\epsilon})D \leq \frac{1}{S} - 1 \leq (1+\bar{\epsilon})D$, we needed $|S-\mu| \leq \frac{\bar{\epsilon}}{4}\mu$.
- So apply the result from the previous slide with $\epsilon = \bar{\epsilon}/4$.
- Need to store $k = \frac{1}{\epsilon^2\delta} = \frac{1}{(\bar{\epsilon}/4)^2\delta} = \frac{16}{\epsilon^2\delta}$ counters.

NOTE ON FAILURE PROBABILITY

$O\left(\frac{1}{\epsilon^2\delta}\right)$ space is an impressive bound:

- $1/\epsilon^2$ dependence cannot be improved.
- No linear dependence on number of distinct elements D.[2]
- But... $1/\delta$ dependence is not ideal. For 95% success rate, pay a $\frac{1}{5\%} = 20$ factor overhead in space.

We can get a better bound depending on $O(\log(1/\delta))$ using <u>exponential tail bounds.</u> We will see next lecture.

[2]Technically, if we account for the bit complexity of storing $S_1, \ldots, S_k$ and the hash functions $h_1, \ldots, h_k$, the space complexity is $O\left(\frac{\log D}{\epsilon^2\delta}\right)$.

DISTINCT ELEMENTS IN PRACTICE

In practice, we cannot hash to real numbers on $[0, 1]$. Instead, map to bit vectors.

Real Flajolet-Martin / HyperLogLog:

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

- Estimate # distinct elements based on maximum number of trailing zeros $\mathbf{m}$.
- The more distinct hashes we see, the higher we expect this maximum to be.

LOGLOG SPACE

Total Space: $O\left(\frac{\log\log D}{\epsilon^2} + \log D\right)$ for an ϵ approximate count.

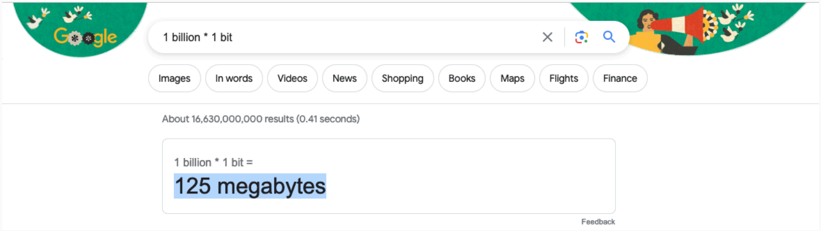
"Using an auxiliary memory smaller than the size of this abstract, the LogLog algorithm makes it possible to estimate in a single pass and within a few percents the number of different words in the whole of Shakespeare's works." – Flajolet, Durand.

Using HyperLogLog to count 1 billion distinct items with 2% accuracy:

$$\begin{aligned}\text{space used} &= O\left(\frac{\log\log D}{\epsilon^2} + \log D\right) \\ &= \frac{1.04 \cdot \lceil \log_2 \log_2 D \rceil}{\epsilon^2} + \lceil \log_2 D \rceil \text{ bits} \\ &= \frac{1.04 \cdot 5}{.02^2} + 30 = 13030 \text{ bits} \approx 1.6\ kB!\end{aligned}$$

HYPERLOGLOG IN PRACTICE

Although, to be fair, storing a dictionary with 1 billion bits only takes 125 megabytes. Not tiny, but not unreasonable.



1 billion * 1 bit

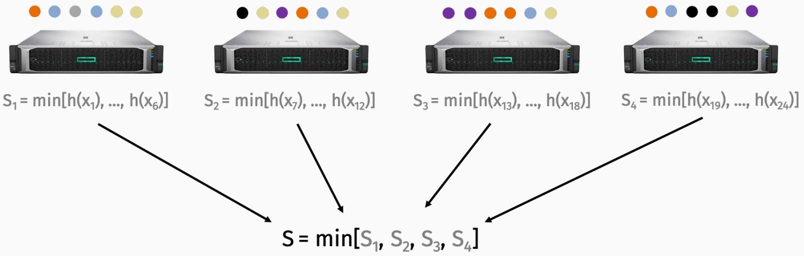
About 16,630,000,000 results (0.41 seconds)

1 billion * 1 bit =
125 megabytes

These estimators become more important when you want to count <u>many</u> different things (e.g., a software company tracking clicks on 100s of UI elements).

DISTRIBUTED DISTINCT ELEMENTS

Also very important in distributed settings.



$S_1 = \min[h(x_1), \ldots, h(x_6)]$
$S_2 = \min[h(x_7), \ldots, h(x_{12})]$
$S_3 = \min[h(x_{13}), \ldots, h(x_{18})]$
$S_4 = \min[h(x_{19}), \ldots, h(x_{24})]$
$S = \min[S_1, S_2, S_3, S_4]$

Distinct elements summaries are “mergeable”. No need to share lists of distinct elements if those elements are stored on different machines. Just share minimum hash value.

HYPERLOGLOG IN PRACTICE

Implementations: Google PowerDrill, Facebook Presto, Twitter Algebird, Amazon Redshift.

Use Case: Exploratory SQL-like queries on tables with 100's of billions of rows.

- Count number of distinct users in Germany that made at least one search containing the word 'auto' in the last month.
- 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 (to estimate rates of spam accounts).

Answering a query requires a (distributed) linear scan over the database: 2 seconds in Google's distributed implementation.

Google Paper: "Processing a Trillion Cells per Mouse Click"