

Hashing and Heavy Hitters

Midterm 2 now graded

I need to process data, calculate stats

at a glance:

word problems a little higher (on average)

MCQ a little lower (on average)

Roadmap for rest of semester

"Average" analysis of algorithms
↑
(w/out sacrificing worst-case robustness)

① Amortized analysis ("average" in hindsight)

② Randomized algorithms ("average" over random bits)

Both are very practical, different perspectives

Tuesday:

② Randomized algorithms ("average" over random bits)

Probability

Events, probabilities,
conditional probabilities,
union bound

Random variables,
expected value,
linearity of expectation

Algorithms

Randomized
min-cut

Quicksort,
Approx. 3-SAT,

Quickselect (PSO)

3-SAT

$$f(x_1, \dots, x_n) = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_2 \vee x_3 \vee x_4) \wedge \dots$$

- NP-Hard to satisfy all the clauses.
- NP-Hard to maximize # satisfied clauses
- Is it hard to satisfy $\frac{OPT}{2}$ clauses?

(OPT = max # satisfied clauses)

Silly $7/8$ -APX

1. sample each $x_i \in \{\text{TRUE}, \text{FALSE}\}$ indep.

$$f(x_1, \dots, x_n) = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_2 \vee x_3 \vee x_4) \wedge \dots$$

satisfied unless $\underbrace{x_1 = F}_{\text{prob } 1/2}$ and $\underbrace{x_2 = F}_{\text{prob } 1/2}$ and $\underbrace{x_3 = T}_{\text{prob } 1/2}$

$$1/2 \times 1/2 \times 1/2 = 1/8$$

$$P[\text{clause satisfied}] = 1 - 1/8 = 7/8$$

$E[\# \text{ clauses satisfied}]$

$$X_i = \begin{cases} 1 & \text{if } i\text{th clause} \\ & \text{satisfied,} \\ 0 & \text{otherwise} \end{cases}$$

Silly $7/8$ -APX

1. sample each $x_i \in \{\text{TRUE}, \text{FALSE}\}$ indep.

$$f(x_1, \dots, x_n) = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_2 \vee x_3 \vee x_4) \wedge \dots$$

satisfied unless $\underbrace{x_1=F}_{\text{prob } 1/2} \text{ and } \underbrace{x_2=F}_{\text{prob } 1/2} \text{ and } \underbrace{x_3=T}_{\text{prob } 1/2}$

$$\frac{1}{2} \times \frac{1}{2} \times \frac{1}{2} = \frac{1}{8}$$

$$P[\text{clause satisfied}] = 1 - \frac{1}{8} = \frac{7}{8}$$

$E[\# \text{ clauses satisfied}]$

$$= E\left[\sum_{i=1}^m X_i\right] = \sum_{i=1}^m E[X_i] = \frac{7}{8} m$$

PCP Theorem

$\forall \epsilon > 0$, it is NP-Hard to get

$(7/8 + \epsilon)$ -APX for 3-SAT

Silly $7/8$ -APX

1. sample each $x_i \in \{\text{TRUE}, \text{FALSE}\}$ indep.

$$f(x_1, \dots, x_n) = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_2 \vee x_3 \vee x_4) \wedge \dots$$

satisfied unless $\underbrace{x_1=F}_{\text{prob } 1/2} \text{ and } \underbrace{x_2=F}_{\text{prob } 1/2} \text{ and } \underbrace{x_3=T}_{\text{prob } 1/2}$

$$1/2 \times 1/2 \times 1/2 = 1/8$$

$$P[\text{clause satisfied}] = 1 - 1/8 = 7/8$$

$$X_i = \begin{cases} 1 & \text{if } i\text{th clause satisfied,} \\ 0 & \text{otherwise} \end{cases}$$

$$E[\# \text{ clauses satisfied}]$$

$$= E\left[\sum_{i=1}^m X_i\right] = \sum_{i=1}^m E[X_i] = \frac{7}{8} m$$

("PCP" = "probabilistically checkable proofs")

Tuesday:

② Randomized algorithms ("average" over random bits)

Probability

Events, probabilities,
conditional probabilities,
union bound

Random variables,
expected value,
linearity of expectation

Algorithms

Randomized
min-cut

Quicksort,
Approx. 3-SAT,

Quickselect (PSO)

Today:

② Randomized algorithms ("average over random bits")

Probability

Hash Functions,
Markov's Inequality

Algorithms

Frequency Estimation,
Heavy Hitters (in streams)

Explore what the world is searching

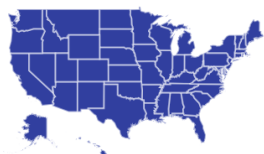
Enter a search term or a topic



Or start with an example

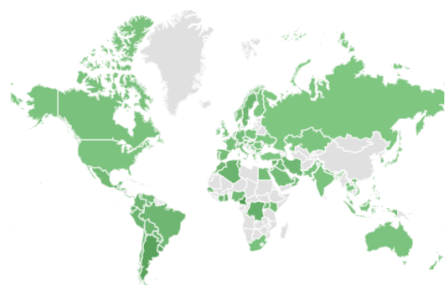
HIDE

● Taylor Swift ● Kim Kardashian



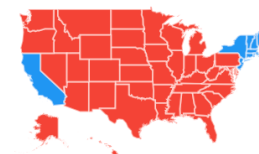
Interest by subregion, Past 7 days, United States

● World Cup



Interest by region, Past 7 days, Worldwide

● Football ● American football



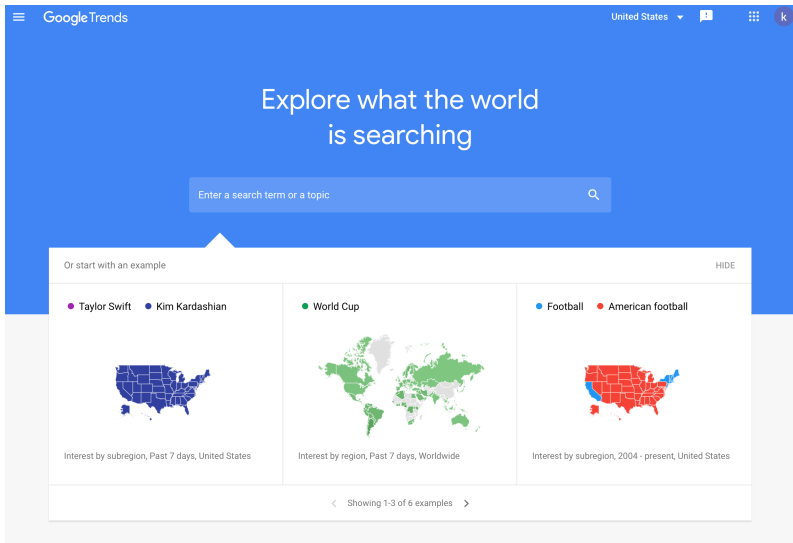
Interest by subregion, 2004 - present, United States

< Showing 1-3 of 6 examples >

Recently trending

Keep up with the latest trending searches

Maundy Thursday 20K+ searches	Dodgers 200K+ searches
Neptune missile 20K+ searches	Liverpool 200K+ searches
Hornets 500K+ searches	Patrick Lyoya 200K+ searches
Pelicans 200K+ searches	Cuba Gooding Jr 100K+ searches
Man City 200K+ searches	Sherri Papini 100K+ searches
MORE TRENDING SEARCHES →	



Recently trending

Keep up with the latest trending searches

Maundy Thursday 20K+ searches	Dodgers 200K+ searches
Neptune missile 20K+ searches	Liverpool 200K+ searches
Hornets 500K+ searches	Patrick Lyoya 200K+ searches
Pelicans 200K+ searches	Cuba Gooding Jr 100K+ searches
Man City 200K+ searches	Sherri Papini 100K+ searches
MORE TRENDING SEARCHES →	

tracking most popular queries is important for
caching

But how?

Way too many different queries to
maintain counters for each

Heavy Hitters In Streams



- Elements one by one online
(can repeat)

a, b, c, b, a, d, a, e, a, b, a, §

- 5% Heavy Hitters: elements that appear $\geq 5\%$ of the time
(more generally 5% $\rightarrow$ parameter $\epsilon > 0$)

Heavy Hitters In Streams

- Elements one by one online
(can repeat)



a, b, c, b, a, d, a, e, a, b, a, f

- 5% Heavy Hitters: elements that appear $\geq 5\%$ of the time

(more generally 5% $\rightarrow$ parameter $\epsilon > 0$)

Goal: identify all heavy hitters
w/ sublinear space

Saving all counts in a dictionary
takes too much space

e.g. $O(\sqrt{n})$, $O(\log n)$ where all elements are
IDs between 1 and n

- 5% Heavy Hitters: elements that appear $\geq 5\%$ of the time

(more generally $5\% \rightarrow$ parameter $\epsilon > 0$)

Goal: identify all heavy hitters
w/ sublinear space

How is this even possible?

(a) Only 20 (5%)-heavy hitters

(b) Randomization: allow small prob. of failure

(c) Approximation: allow some margin of error

rigorous, worst-case bounds



Today:

② Randomized algorithms ("average over random bits")

Probability

Hash Functions,
Markov's Inequality

Algorithms

Frequency Estimation,
Heavy Hitters (in streams)

Markov's Inequality (special case)

Let $X \geq 0$ be nonnegative RV

$$\Pr[X \geq 2E[X]] \leq \frac{1}{2}$$

e.g.

$\leq 50\%$ of US popul. earns $2 \times$ average

$\leq 50\%$ of NBA players score
 $2 \times$ league average points

$\leq 50\%$ students get
 $2 \times$ average midterm score

Markov's Inequality

Let $X \geq 0$ be nonnegative RV

$$\Pr[X \geq 2E[X]] \leq \frac{1}{2}$$

Proof

$$E[X] = \sum_x P[X=x] \cdot x$$

$$\geq \sum_{x: x \geq 2E[X]} P[X=x] \cdot x$$

$$\geq (2E[X]) \sum_{x: x \geq 2E[X]} P[X=x]$$

$$= 2E[X] \Pr[X \geq 2E[X]]$$

Markov's Inequality

Let $X \geq 0$ be nonnegative RV

$$\Pr[X \geq 2E[X]] \leq \frac{1}{2}$$

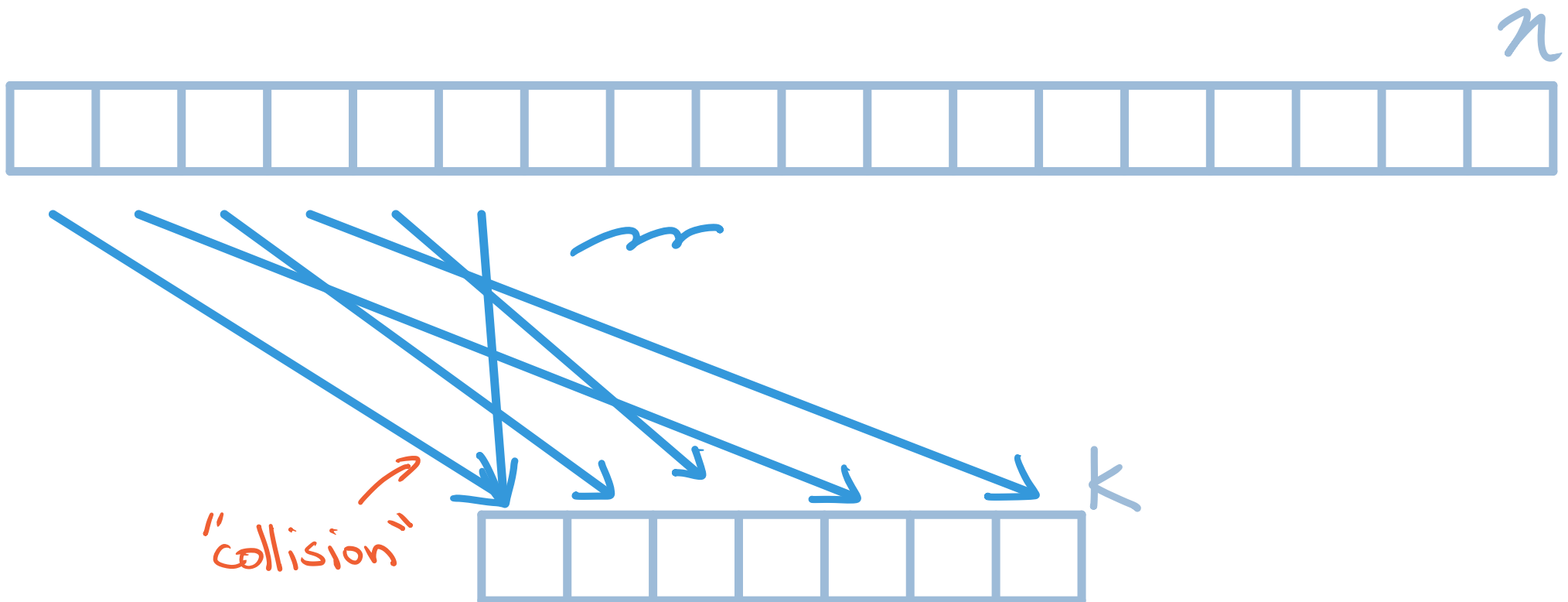
more generally, for $\alpha \geq 1$,

$$\Pr[X \geq \alpha E[X]] \leq \frac{1}{\alpha}$$

② Hash Functions

Randomly constructed function

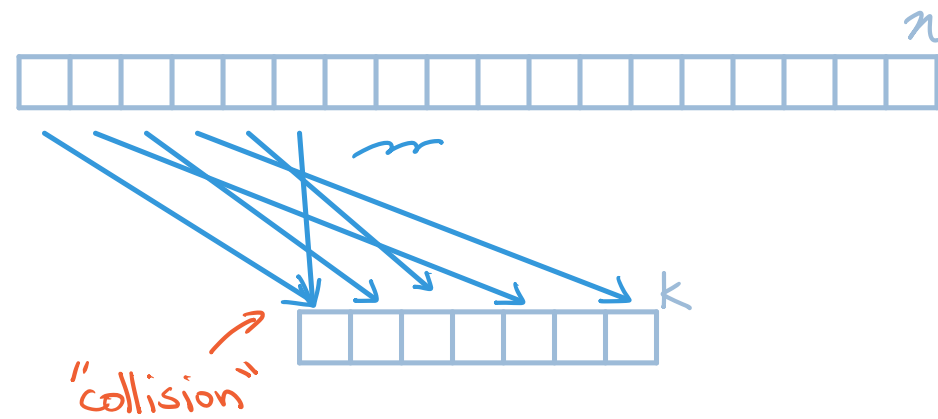
$h: [n] \rightarrow [k]$
big $\nearrow$ $[n]$ $\nwarrow$ $[k]$ rel. small



② Hash Functions

Randomly constructed function

$h: [n] \rightarrow [k]$
big $\rightarrow$ rel. small



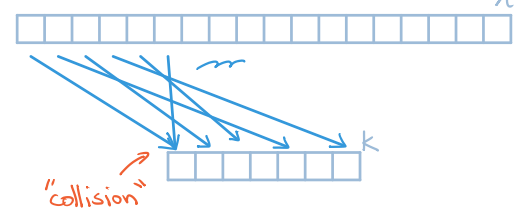
"Ideal Hash Function"

$h: [n] \rightarrow [k]$

w/ each $h(i)$ indep. unif. random in $[k]$

requires $O(\log(k^n)) = O(n \log k)$ space $\ddot{\smile}$

Universal Hash Functions



$h: [n] \rightarrow k$ s.t. for all pairs $i, j \in [n]$,

$$\Pr[h(i) = h(j)] \leq \frac{1}{k}$$

(much weaker than ideal hash)

e.g. $h(x) = ((ax + b) \bmod p) \bmod k$

where

- p prime, $> k$
- a in $\{1, \dots, p\}$ unif. random
- b in $\{0, \dots, p-1\}$ unif. random

(proofs left as exercise)

2 new probabilistic tools

① "Markov's Inequality"

$$\Pr[X \geq 2E[X]] \leq \frac{1}{2}$$

② universal hash $h: [n] \rightarrow [k]$

$$\Pr[h(i) = h(j)] \leq \frac{1}{k}$$

Today:

② Randomized algorithms ("average" over random bits)

Probability

Hash Functions,

Universal Hash Functions

$h: [n] \rightarrow k$ s.t. for all pairs $i, j \in [n]$,
 $\Pr[h(i) = h(j)] \leq \frac{1}{k}$ (much weaker than ideal hash)

e.g. $h(x) = (ax + b) \bmod p \bmod k$

where • p prime, $> k$

• a in $\{1, \dots, p\}$ unif. random

• b in $\{0, \dots, p-1\}$ unif. random



Markov's Inequality

Markov's Inequality (special case)

Let $X \geq 0$ be nonnegative RV

$$\Pr[X \geq 2E[X]] \leq \frac{1}{2}$$

Algorithms

Frequency Estimation,
Heavy Hitters (in streams)

Heavy Hitters In Streams



- Elements one by one online
(can repeat)

a, b, c, b, a, d, a, e, a, b, a, §

- 5% Heavy Hitters: elements that appear $\geq 5\%$ of the time
(more generally 5% $\rightarrow$ parameter $\epsilon > 0$)

Heavy Hitters In Streams

- Elements one by one online
(can repeat)



a, b, c, b, a, d, a, e, a, b, a, f

- 5% Heavy Hitters: elements that appear $\geq 5\%$ of the time

(more generally 5% $\rightarrow$ parameter $\epsilon > 0$)

Goal: identify all heavy hitters
w/ sublinear space

Saving all counts in a dictionary
takes too much space

e.g. $O(\sqrt{n})$, $O(\log n)$ where all elements are
IDs between 1 and n

From now on:

ϵ -heavy hitters for param. $\epsilon > 0$

New goal:

estimate relative frequency of each element
(up to ϵ additive error)

absolute freq (x) = # times x appears in stream

$$\left(\begin{array}{c} \text{relative} \\ \text{freq. of } x \end{array} \right) = \frac{(\text{abs. freq. } x)}{\text{total length of stream}}$$

ϵ -heavy hitter $\Leftrightarrow$ relative freq. $\geq \epsilon$

So to recap:

want to estimate all relative frequencies
up to ϵ -additive error

not enough space for all counters

lucky for us:

0 is a good estimate for all but $\frac{1}{\epsilon}$ elements

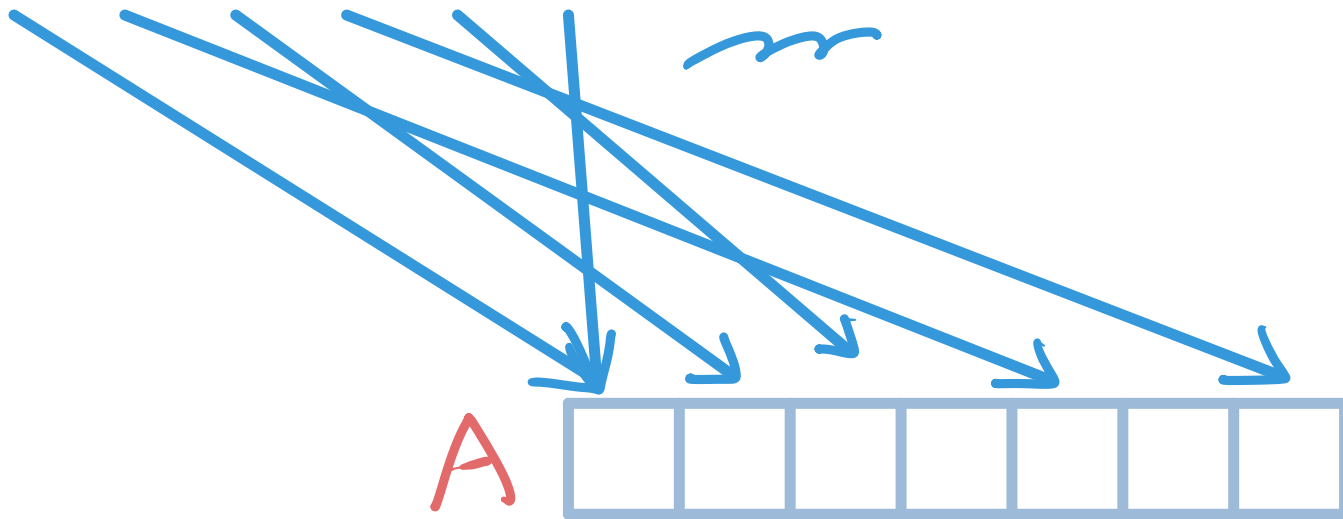
still: don't know which $\frac{1}{\epsilon}$ elements to count

Washing has enter the chat

Idea: allocate $w = 10/\epsilon$ counters

hash element i to counter $h(i)$

n



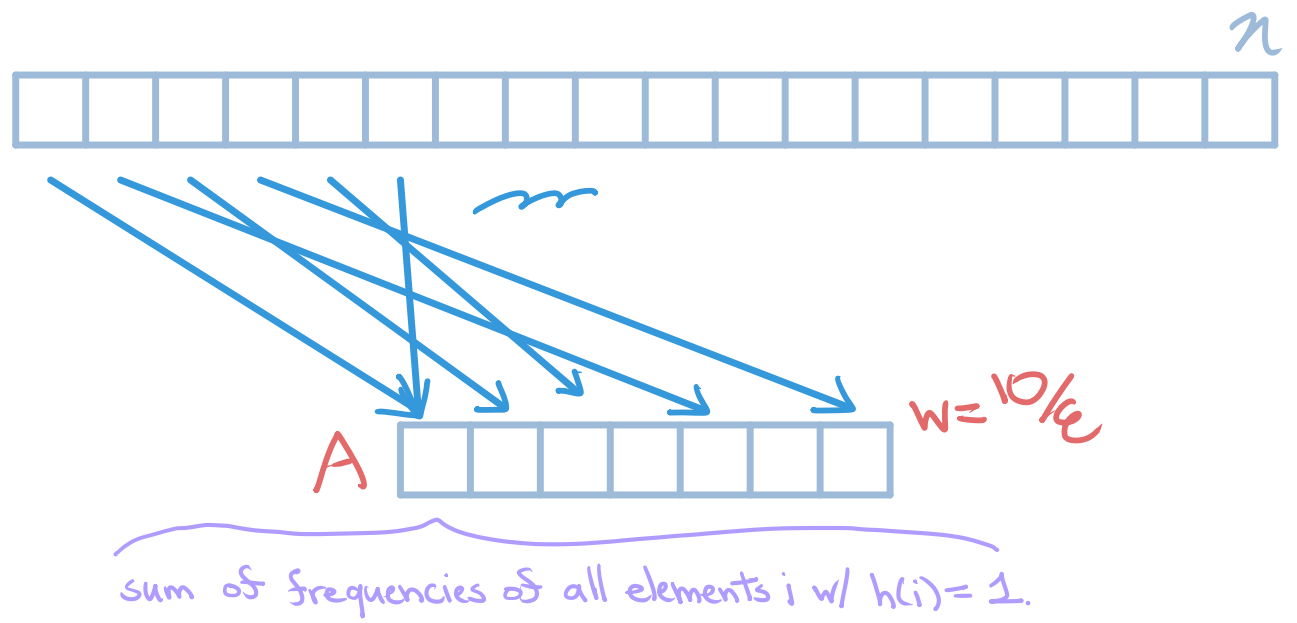
$w = 10/\epsilon$

sum of frequencies of all elements i w/ $h(i) = 1$.

To estimate frequency f_i of i

return $h(i)$

↑
overcounts f_i



Notation

$A[1..w]$ = array of counters

f_i = (true) frequency of i

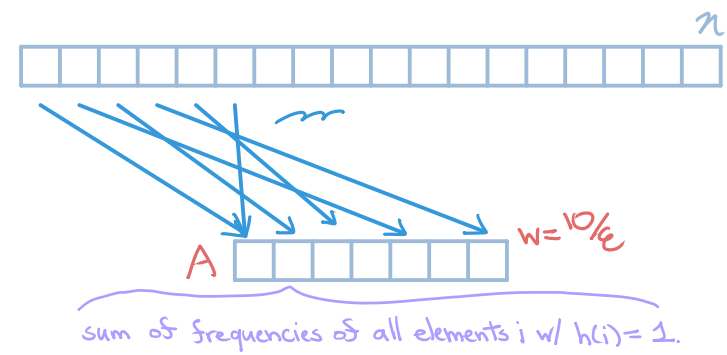
$h: [n] \rightarrow [w]$: universal hash function

m = length of stream

e.g. $A[j] = \sum_{i: h(i)=j} f_i$

Lemma For all i ,
w/ probability $\geq \frac{1}{2}$,

$$f_i \leq A[h(i)] \leq f_i + \frac{\epsilon}{5} m$$



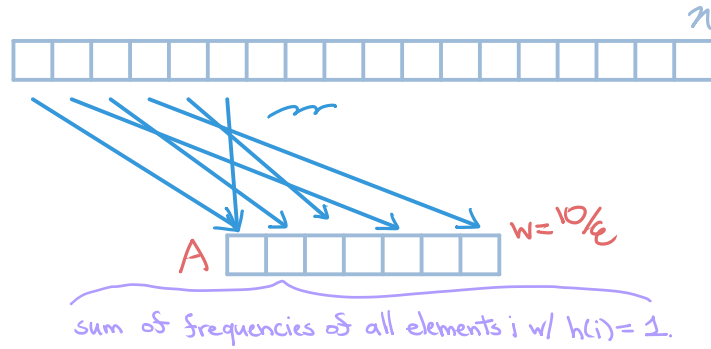
$A[1..w]$ = array of counters

f_i = (true) frequency of i

$h: [n] \rightarrow [w]$: universal hash fn.

m = length of stream

Recap:



$A[1..w]$ = array of counters

f_i = (true) frequency of i

$h: [n] \rightarrow [w]$: universal hash fn.

m = length of stream

Lemma For all i , w/ probability $\geq \frac{1}{2}$,
 $f_i \leq A[h(i)] \leq f_i + \frac{\epsilon}{5} m$

we have: For each element, good estimate of frequency w/ constant prob.

we want: estimates of all frequencies w/ very high probability

we want to "amplify" our bounds

Repeatedly flip coins:

$$P[\text{first toss is tails}] = \frac{1}{2}$$

$$P[\text{first 2 tosses are tails}] = \frac{1}{4}$$

6

$$P[\text{first 100 tosses are tails}] = \frac{1}{2^{100}}$$

make d copies of hashed counters

A_1, h_1

--	--	--	--	--	--	--	--

A_2, h_2

--	--	--	--	--	--	--	--

A_3, h_3

--	--	--	--	--	--	--	--

A_4, h_4

--	--	--	--	--	--	--	--

⋮

A_d, h_d

--	--	--	--	--	--	--	--

total space
 $O(wd)$

estimate f_i w/ $\min_{j=1, \dots, d} A_j[h_j(i)]$

bad
estimate

iff

ALL
hashed counters
are bad

prob $1/2^d$

estimate $\sum_i$

$$\text{w/ } \min_{j=1, \dots, d} A_j[h_j(i)]$$

bad
estimate

iff ALL
hashed counters
are bad

prob $1/2^d$

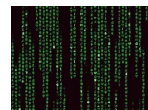
$$\text{set } d = 10 \log(n)$$

each element is "bad" w/ prob.

$$\leq 2^d = \frac{1}{n^{10}}$$

goal: ensure all elements are correct

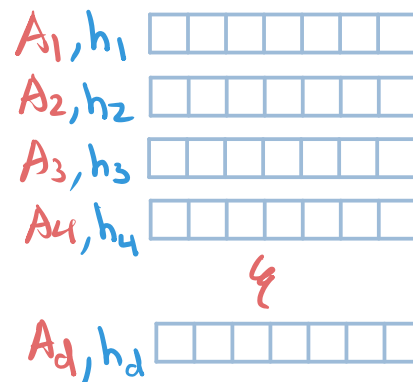
Heavy Hitters In Streams



- Elements one by one online
(can repeat)

a, b, c, b, a, d, a, e, a, b, a, f

- 5% Heavy Hitters: elements that appear $\geq 5\%$ of the time
(more generally $5\% \rightarrow$ parameter $\epsilon > 0$)



estimate $\sum_i$

$$\text{w/ } \min_{j=1, \dots, d} A_j[h_j(i)]$$

bad
estimate

iff ALL
hashed counters
are bad
prob $1/2^d$

$\Rightarrow$ all elements estimated accurately w/ prob. $\geq 1 - \frac{1}{n^9}$

Space:

$$O(wd) = O(\log(n)/\epsilon)$$

Heavy Hitters In Streams



- Elements one by one online (can repeat)

a, b, c, b, a, d, a, e, a, b, a, f

- 5% Heavy Hitters: elements that appear $\geq 5\%$ of the time

(more generally 5% $\rightarrow$ parameter $\epsilon > 0$)

A_1, h_1

A_2, h_2

A_3, h_3

A_4, h_4

$\vdots$

A_d, h_d

estimate S_i

w/ $\min_{j=1, \dots, d} A_j[h_j(i)]$

bad estimate iff ALL hashed counters are bad
 $\uparrow$
 prob $\frac{1}{2^d}$

set $d = 10 \log(n)$

each element is "bad" w/ prob.

$$\leq 2^{-d} = \frac{1}{n^{10}}$$

$$\text{Prob[any element bad]} \leq \sum_{i=1}^n \text{Pr[element } i \text{ bad]} \\ \leq n \cdot \frac{1}{n^{10}} = \frac{1}{n^9}$$

$\Rightarrow$ all elements estimated accurately w/ prob. $\geq 1 - \frac{1}{n^9}$

$\Rightarrow$ all elements estimated accurately w/ prob. $\geq 1 - \frac{1}{n^9}$

Space: $O(wd) = O(\log(n)/\epsilon)$

All put together

Estimate all rel. frequencies up to $\pm \epsilon$ error w/

- $O(\log n / \epsilon)$ space

- $\geq 1 - \frac{1}{n^{10}}$ prob. of success

Heavy Hitters In Streams



- Elements one by one online (can repeat)

a, b, c, b, a, d, a, e, a, b, a, f

- 5% Heavy Hitters: elements that appear $\geq 5\%$ of the time

(more generally 5% $\rightarrow$ parameter $\epsilon > 0$)

A_1, h_1

A_2, h_2

A_3, h_3

A_4, h_4

$\vdots$

A_d, h_d

estimate f_i

w/ $\min_{j=1, \dots, d} A_j[h_j(i)]$

bad estimate iff ALL hashed counters are bad
 $\uparrow$
 prob $\frac{1}{2^d}$

set $d = 10 \log(n)$

each element is "bad" w/ prob.

$$\leq 2^{-d} = \frac{1}{n^{10}}$$

$$\text{Prob[any element bad]} \leq \sum_{i=1}^n \text{Pr[element } i \text{ bad]} \\ \leq n \cdot \frac{1}{n^{10}} = \frac{1}{n^9}$$

$\Rightarrow$ all elements estimated accurately w/ prob. $\geq 1 - \frac{1}{n^9}$

Today:

② Randomized algorithms ("average over random bits")

Probability

Hash Functions,
Markov's Inequality

Algorithms

Frequency Estimation,
Heavy Hitters (in streams)

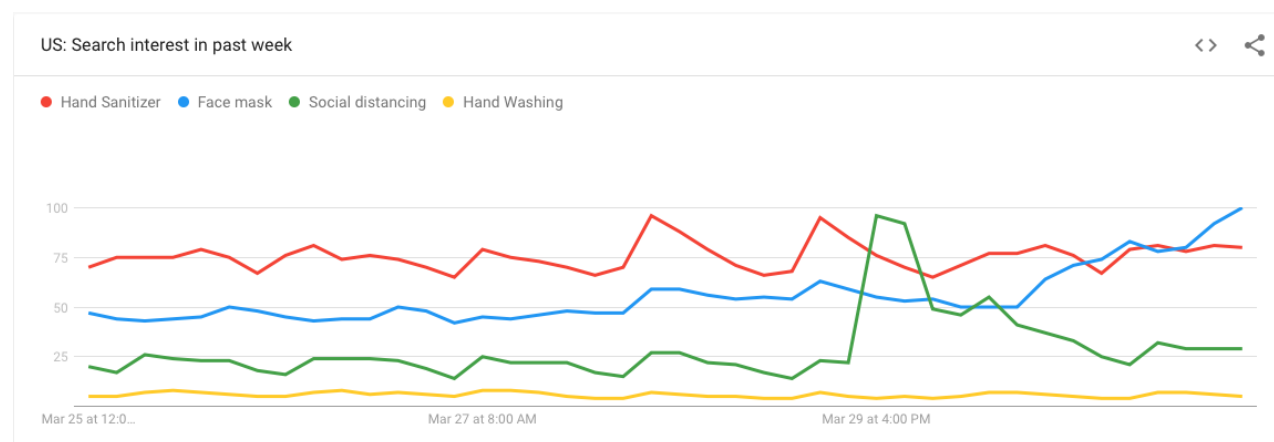
next time: hash tables

Chapter 22

Hashing and Heavy Hitters

22.1 Trends

Google has an interesting web page called “[Google trends](#)”, which tracks surging search queries around the world in real time. In Spring 2021, there was even a subpage of search trends specifically related to the Covid-19.



Google tracks the trending search queries not just for the sake of curiosity. Its goal is not only to serve queries, but to serve queries *fast*. The best way to serve something quickly is to have it ready before it is even asked for. By keeping track of the ‘heavy hitter’ search terms - a few search terms that make up a disproportionate amount of the search traffic - Google can cache the answers to most search requests before they are even made.

Google currently serves billions¹ of queries a day. Given the sheer magnitude of Google’s search traffic, and the diversity of search queries, it is not obvious

¹Maybe 7 billion? See <https://www.internetlivestats.com/google-search-statistics/>

how to identify the most popular search queries. Certainly one cannot simply have a tally for each search term, since there are too many search terms out there to be stored. More generally, it is prohibitively expensive to maintain any data structure proportional to the input size. Somehow we need an approach that takes *sublinear* space.

22.2 Streaming

We study the heavy hitter problem in the **streaming model** of computation. In the streaming model, the input is a sequence of items presented to the algorithm *one at a time*. The algorithm cannot simply write down everything and solve the problem offline, because it is restricted to using very little space compared to the input size. For example, if there are n items in the stream, then the algorithm might be restricted to $n^{1/4}$ space, or even better, $O(\log n)$ space. Because the space is so much smaller than the input size, each time an item from the stream is given to the algorithm, the algorithm needs to be fairly selective about what parts of the item (if any) it wants to store.



We formalize the heavy hitters problem as follows. We have n elements in a stream $e_1, \dots, e_n$, where each element is from some large universe $\{1, \dots, m\}$. Elements can repeat. The **absolute frequency** of an element e , denoted f_e , is the number of times the element appears in the stream. The **relative frequency**, denoted p_e , is the fraction of the stream that the element appears in. In a stream of n elements, the relative frequency of an element is the total frequency divided by n .

Ideally we could keep track of the exact frequencies of all the elements. This is impossible with less than $\min\{m, n\}$ bits². Here, and unlike standard algorithmic settings, $O(n)$ or $O(m)$ bits is not good enough. *We have entered an algorithmic regime where simple and exact computations cannot be taken for granted.* Our first step, then, is to identify some new problems that are both *tractable* and *useful*. We may have to relinquish exactness and consider *approximations*, where we allow for some error that we can analyze and control. And, last but not least, we will give up on deterministic computation, and design *randomized* algorithms that have some small – analyzed and controlled – *probability of failure*.

² One can formalize this impossibility as follows. There are $\binom{n+m-1}{m-1}$ ways to make a frequency

ϵ -heavy hitters. Given a fixed parameter $\epsilon \in (0,1)$, an **ϵ -heavy hitter** is an element with relative frequency $\geq \epsilon$. The heavy hitters problem is to identify all of the ϵ -heavy hitters for an input parameter $\epsilon > 0$. Note that there can only be $(1/\epsilon)$ -many ϵ -heavy hitters, which preserves some hope that we can identify all of them with space proportional to $1/\epsilon$, rather than n .

We also consider a closely related problem of **(approximate) frequency estimation**. Given a fixed error parameter $\epsilon > 0$, the goal is to estimate every element's relative frequency up to an additive error of ϵ . Equivalently, we want to estimate the absolute frequency of each element up to an ϵn -additive factor. At first it might seem impossible to estimate n -many counts with $o(n)$ space. We have already argued that n exact counters is impossible. Allowing for ϵ -relative error, however, means that o is a satisfactory estimate for all but at most $\lfloor 1/\epsilon \rfloor$ elements. That is, the room for approximation

If we can estimate the absolute frequency of each element up to additive error ϵ , then one can find all (3ϵ) -heavy hitters by considering all of the elements with estimated value at least 2ϵ . Such a list might also include $1/\epsilon$ extra elements who have relative frequency $< 3\epsilon$, but still have frequency $> 1/\epsilon$. Frequency estimation gives more information than just who are the heavy hitters. By knowing their frequencies up to some small error, one can also rank them (approximately) from most to least frequent, such as in Google trends.

Conversely, if we knew *a priori* which of the elements are the ϵ -heavy hitters, then ϵ -frequency estimation is trivial. Namely, we would maintain a counter for each of the $\lfloor 1/\epsilon \rfloor$ -many ϵ -heavy hitters. All other elements are ignored and assigned frequency 0. Of course this approach is not possible since we do not know the heavy hitters. Surprisingly we will pursue a strategy that is actually quite similar. We will allocate $w = O(1/\epsilon)$ counters, hoping to use one counter for each heavy hitter (plus a few extra for safe measure). Although we do not know the heavy hitters, we will use *randomized hash functions* to obtain a similar effect.

vector of m items where the total sum is n . (*Why?*) Suppose we claim that we can describe any combination of counts with k bits. Each k -bit string can describe by only one of these $\binom{n+m-1}{m-1}$ outcomes, so we must have $2^k \geq \binom{n+m-1}{m-1}$, hence $k \geq \log \left(\binom{n+m-1}{m-1} \right)$. Here $\log$ denotes $\log_2$. We also have

$$\binom{m+n-1}{m-1} = \left(\frac{m+n-1}{m-1} \right)^{m-1} \text{ and also } \binom{m+n-1}{m-1} = \binom{m+n-1}{n} \geq \left(\frac{m+n-1}{n} \right)^n$$

Thus

$$k \geq (m-1) \log \left(1 + \frac{n}{m-1} \right) \text{ and } k \geq n \log \left(1 + \frac{m-1}{n} \right).$$

Slightly better low bounds can be obtained via *Stirling's approximation*, which is also related to entropy.

22.3 New Probabilistic Tools

We will need two basic and useful probabilistic tools: hash functions, and Markov's inequality.

22.3.1 Hashing

Likely the reader has used hash tables before, and may be aware that they use *hash functions* to randomly map keys to slots in an array. We will discuss hash tables in detail in the following chapter. Let us now define hash functions mathematically.

Loosely speaking, a hash function is a randomly constructed function $h : [m] \rightarrow [k]$ where the values $h(i)$ are (in a qualified sense) randomly distributed through k . A **collision** is a pair of distinct indices $i_1 \neq i_2 \in [m]$ such that $h(i_1) = h(i_2)$. In most applications, m is much (much, much) larger than k . In this case, there are necessarily many “collisions” (see exercise 23.1). A goal of hash functions is to distribute these collisions “fairly”.

Ideal hash functions. One way to construct a hash function, for example, is to sample, for each $i \in [m]$, a value $h(i) \in [k]$ independently and uniformly at random. This produces an “ideal hash function”, defined as follows.

Definition 22.1. *An ideal hash function $h : [m] \rightarrow [k]$ is a uniformly random function $h : [m] \rightarrow [k]$. That is, each $h(i)$ is drawn from $[k]$ independently uniformly at random.*

An ideal hash function h is particularly easy to reason about. For example, for every input i and possible output $j \in [k]$, we have

$$\mathbf{P}[h(i) = j] = \frac{1}{k}.$$

More generally, for any ℓ distinct inputs $i_1, \dots, i_\ell \in [m]$ and ℓ possible outputs $j_1, \dots, j_\ell \in [k]$, we have

$$\begin{aligned} & \mathbf{P}[h(i_1) = j_1, h(i_2) = j_2, \dots, h(i_\ell) = j_\ell] \\ &= \mathbf{P}[h(i_1) = j_1] \mathbf{P}[h(i_2) = j_2] \cdots \mathbf{P}[h(i_\ell) = j_\ell] \\ &= \frac{1}{k^\ell}. \end{aligned}$$

Ideal hash functions are a good model to keep in mind when designing randomized algorithms. Assuming the hash values are completely independent simplifies calculations. In reality, however, ideal hash functions are very expensive to make and store. Indeed, one has to have $m \log k$ bits to be able to describe all of the possible functions from $[m]$ to $[k]$ (as there are k^m such functions, and we must

pay the logarithm of this quantity). This is particularly ill-suited to our streaming setting where m is astronomical and the goal is to use space sublinear to the input size.

Universal hash functions. Fortunately, in most applications, only a *limited amount of randomization* is actually required. For the current discussion, we only require “universal” hash functions that have “ideal pairwise collision probabilities”, in the following sense.

Definition 22.2. A random hash function $h : [m] \rightarrow [k]$ is **universal** if for any distinct indices $i_1 \neq i_2 \in [m]$, we have

$$\mathbf{P}[h(i_1) = h(i_2)] = \frac{1}{k}.$$

In contrast to ideal hash functions, universal hash functions can be constructed compactly by the following simple algorithm.

new-universal-hash-function(m, k)

1. Pick a prime number p greater than m .
2. Draw an integer $a \in \{1, \dots, p-1\}$ uniformly at random.
3. Draw an integer $b \in \{0, \dots, p-1\}$ uniformly at random.
4. Return the function $h(x) = ((ax + b) \bmod p) \bmod k$.

Theorem 22.3. Consider the randomly constructed function $h : [m] \rightarrow [k]$

$$h(x) = (ax + b \bmod p) \bmod k,$$

where p is a prime number larger than m , $a \in \{1, \dots, p-1\}$ is drawn uniformly at random, and $b \in \{0, \dots, p\}$ is drawn uniformly at random. Then h is a 2-universal hash function.

The proof of theorem 22.3 is given as exercise 22.3. Here we will assume theorem 22.3 and focus on its application to the heavy hitters problem.

22.3.2 Markov’s inequality

Consider the following fact.

Fact 22.4. Let $X \geq 0$ be a nonnegative random variable. Then

$$\mathbf{P}[X \geq 2\mathbf{E}[X]] \leq \frac{1}{2}.$$

Fact 22.4 is intuitive and obvious – at least, from a counterfactual perspective: a nonnegative random variable cannot be more than two times its average more than half the time. For example, no more than 50% of the population can make twice as much as the the average income. No more than 50% of NBA players can have a scoring average more than twice the league-wide average. Another way to put it: if a nonnegative random variable has value ≥ 10 at least half of the time, then *of course* its average is ≥ 5 .

A full proof of **Fact 22.4** is as follows.

$$\begin{aligned} \mathbf{E}[X] &\stackrel{(a)}{=} \sum_x \mathbf{P}[X = x] \cdot x \stackrel{(b)}{\geq} \sum_{x: x \geq 2\mathbf{E}[X]} \mathbf{P}[X = x] \cdot x \\ &\stackrel{(c)}{\geq} 2\mathbf{E}[X] \sum_{x: x \geq 2\mathbf{E}[X]} \mathbf{P}[X = x] \stackrel{(d)}{=} 2\mathbf{E}[X] \mathbf{P}[X \geq 2\mathbf{E}[X]]. \end{aligned}$$

Here (a) is by definition of $\mathbf{E}[X]$. (b) drops some of the nonnegative values where $x < 2\mathbf{E}[X]$. (c) is because all the values x remaining in the sum are $\geq 2\mathbf{E}[X]$. (d) is by definition of $\mathbf{P}[X \geq 2\mathbf{E}[X]]$. Rearranging the inequality gives the desired result.

By replacing 2 with arbitrary $\alpha > 1$, one obtains the more general **Markov's inequality**.

Theorem 22.5 (Markov's inequality). *Let $X \geq 0$ be a nonnegative random variable, and let $\alpha > 1$. Then*

$$\mathbf{P}[X \geq \alpha \mathbf{E}[X]] \leq \frac{1}{\alpha}.$$

The reader is asked to prove Markov's inequality (for general α) in exercise 22.2. (Hint: one can follow the outline of the proof of **Fact 22.4** above.)

22.4 Using hashing to approximate frequencies

Let us now return to the frequency estimation problem. We have elements from the set $[m]$ coming in a stream of n total elements. The goal is to estimate the absolute frequency of each element up to an ϵn -additive factor. The crux of the problem is that we only want our total space usage to be (more or less) independent of the length of the stream, n , or the number of elements, m . We mentioned briefly above that if we knew the heavy hitters, then we could just maintain a counter for each one. Since there are at most $1/\epsilon$ heavy hitters, this approach would satisfy our space constraints. Of course we do not know the heavy hitters. In the following, we will use hash functions to, in effect, *guess* the heavy hitters.

We first create an array of counters $A[1..w]$ with $w = \lceil 2/\epsilon \rceil$ entries. Note that $2/\epsilon$ is extremely small compared to the total length of the stream, or the distinct number of keys. We also sample a universal hash function $h : \{1, \dots, m\} \rightarrow \{1, \dots, w\}$. For each element e presented by the stream, we increase $A[h(e)]$ by 1. In turn, for each element e , we treat $A[h(e)]$ as an estimate for f_e .

hashed-counters($\epsilon > 0$)

1. allocate an array of size $A[1..w]$ for $w = \lceil 2/\epsilon \rceil$
2. sample a universal hash function $h : \{1, \dots, m\} \rightarrow \{1, \dots, w\}$
3. for each item e in the stream
 - A. $A[h(e)] \leftarrow A[h(e)] + 1$

$A[h(e)]$ never underestimates f_e , and the hope is that it does not overestimate f_e by too much. The risk of error comes from other elements' frequencies possibly adding more than ϵn to $A[h(e)]$. Here the intuition is that the “noise” coming from other frequencies is spread out by the hash function over $\epsilon/2$ entries, so we would only expect $\epsilon n/2$ error for each element e . To translate “expected error” to “probability of error”, we use Markov's inequality, as follows.

Lemma 22.6. *For each element e , with probability $\geq 1/2$, we have*

$$f_e \leq A[h(e)] \leq f_e + \epsilon n.$$

Proof. We have $A[h(e)] \geq f_e$ always because $A[h(e)]$ is a sum of frequencies of elements with hash code $h(e)$, which of course includes e . The expected additive error is bounded above by

$$\mathbf{E}[A[h(e)]] - f_e \stackrel{(a)}{=} \sum_{d \neq e} f_d \mathbf{P}[h(d) = h(e)] \stackrel{(b)}{\leq} n/w \leq \frac{\epsilon}{2} n. \quad (22.1)$$

Here (a) is by linearity of expectation. (b) is because h is universal. Now we have

$$\mathbf{P}[A[h(e)] \geq f_e + \epsilon n] \stackrel{(c)}{\leq} \mathbf{P}[A[h(e)] - f_e \geq 2 \mathbf{E}[A[h(e)] - f_e]] \stackrel{(d)}{\leq} \frac{1}{2}$$

Here (c) plugs in the inequality obtained in (22.1). (d) applies Markov's inequality, where we note that $A[h(e)] - f_e \geq 0$. ■

Given that it is impossible to track frequencies exactly in sublinear space, it is surprising that we can now count every element's frequency with extremely small space, *sometimes and with small additive error*. The algorithm, including the construction of the universal hash function, is extremely simple.

22.5 Amplification

section 22.4 shows how to estimate each element with fairly small error with constant probability of error. Our goal now is to reduce the error probability enough to even take the union bound over all of the elements, and thus estimate all frequencies up to ϵn -additive error.

The idea is to use repetition, and one analogy is coin tossing. The goal is to flip enough coin tosses to get at least one heads with very high probability. With one coin toss, the probability that it is tails is $1/2 = .5$. With two coin tosses, the probability that both come up tails is still $1/4 = .25$. But with 100 coin tosses, the probability that all 100 coin tosses come up tails is $1/2^{100} \approx .0000000000000000000000000000007886....$

The point is that independent trials magnify the probability of at least one success exponentially. For a specified probability of error $\delta \in (0,1)$, the algorithm count-min-sketch below makes $\lceil \log 1/\delta \rceil$ independent instances of hashed-counters(ϵ). For each element e , it uses the minimum estimate over all of the instances of hashed-counters. The overall data structure fails for an element e only if *every* instance of hashed-counters fails, which by the analogy with coins, is exceedingly unlikely.

count-min-sketch($\epsilon > 0, \delta > 0$)

1. build $d = \lceil \log 1/\delta \rceil$ instances $(A_1, h_1), \dots, (A_d, h_d)$ of $\text{hashed-counters}(\epsilon)$ over the stream
2. to query an element e :

A. return $\min_{i=1,\dots,d} A_i[h_i(e_i)]$

Lemma 22.7. *For each element e , with probability $\geq 1 - \delta$, we have*

$$\min_{i=1,\dots,d} A_i[h_i(e)] \leq f_e + \epsilon n.$$

Proof. We have

$$\mathbf{P}\left[\min_{i=1,\dots,d} A_i[h_i(e)] > f_e + \epsilon n\right] \stackrel{(a)}{=} \prod_{i=1}^d \mathbf{P}[A_i[h_i(e)] > f_e + \epsilon n] \stackrel{(b)}{\leq} \frac{1}{2^d} \leq \delta.$$

Here (a) is by independence of each $A_i[h_i(e)]$. (b) is by lemma 22.6.

For δ set to a polynomial of $1/n$, the probability of error becomes low enough to take a union bound over all elements, as follows.

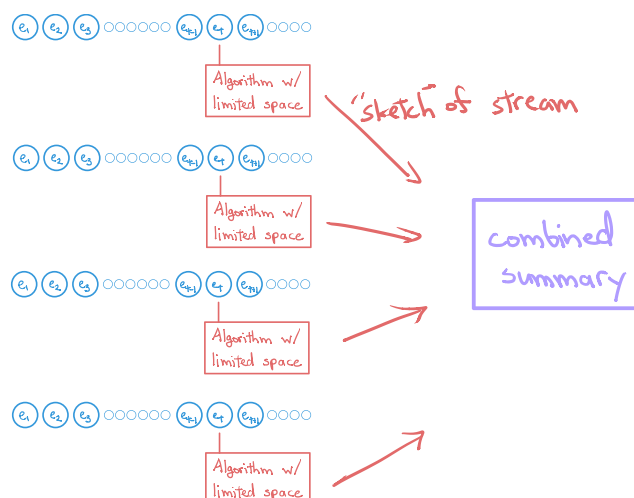
Theorem 22.8. *With probability at least $1 - 1/n$, over a stream of length n , count-min-sketch($\epsilon, 1/n^2$) overestimates the total frequency of each element with additive error at most ϵn and total space $O(\log(n)/\epsilon)$.*

Proof. By lemma 22.7, we have probability of error $\leq 1/n^2$ for each element e . Taking the union bound over all n elements in the stream, we have probability of error $\leq 1/n$. ■

22.6 Extensions

22.6.1 Crossing streams

One can extend the streaming model to multiple streams in the following *distributed* model of computation. Here we have several streams simultaneously, each served by an algorithm using sublinear space. The goal is to solve the heavy hitters problem over the *combined streams*.



count-min-sketch has the convenient property of being a **sketch**. To handle multiple streams, we have an instance of count-min-sketch for each stream arranged so that they are *all using the same hash functions*. To combine their results, we simply sum up the arrays A_i of hashed sums entry-wise. The result is an instance of count-min-sketch over the combined streams.

22.6.2 Turnstile streams

Consider the more general model where each item in the stream consists of an element e and a value Δ , signifying that we should increase the frequency count for e , f_e by Δ . Δ is allowed to be negative, with the restriction that the frequency f_e of each element (which is now the sum of Δ 's for that element) remains nonnegative. This model is sometimes called a “turnstile stream”, in the

sense that a turnstile counting the number of people in an amusement park is always nonnegative because each decrease corresponds to a person who entered the park earlier.

count-min-sketch adapts immediately to turnstile stream, by simply adding Δ to $A_i[h_i(e)]$ for each instance (A_i, h_i) of hashed-counters. The additive error is now ϵ times the sum of all Δ 's in the stream.

22.7 Takeaways

- There are many basic and useful problems – heavy hitters with sublinear space being on of our first examples – that are too difficult or even impossible to compute exactly and deterministically. Instead we consider *randomized approximation algorithms* that are potentially more scalable. This requires *quantitative analysis* to address the approximation factor in addition to algorithm design.
- count-min-sketch uses *hashing* to try to distribute the heavy hitters across an array. It does not know which are the heavy hitters, but relies on *randomization* to separate the heavy hitters (most of the time) in an *oblivious* fashion.
- *Ideal hash functions*, while easy to reason about, are prohibitively expensive. Luckily, weaker hash functions with *limited randomness* often suffice, and are easily constructed. count-min-sketch requires only *universal hash functions*. Universal hash functions can be implemented very easily.
- *Linearity of expectation*, combined with universal hash functions, implies that the noise seen by a particular element is evenly spread out *on average*. *Markov's inequality* allowed us to argue that the noise encountered by an element is close to the average, most of the time.
- count-min-sketch *amplifies* the probability of success by taking the minimum over many independent trials. A particular element is miscounted if and only if *all* independently trials miscount the element, which happens with vanishingly small probability.
- The error probability drops so rapidly that we can apply the *union bound* over all of the elements after just $O(\log n)$ trials.
- count-min-sketch does not give *unbiased estimates* of the counters. Instead, count-min-sketch tries to be *within a prescribed error with high probability*. It is *consistent*. It is more important to be *consistent* than *unbiased*, since we

can (psychologically) adjust for the bias. Many real-world apparatus are designed on this principle.

22.8 Exercises

Exercise 22.1. Let $h : [m] \rightarrow [k]$ be *any* fixed function.

1. Prove that the number of collisions is

$$\geq \frac{m(m-n)}{2n}$$

2. Show that the above inequality is tight when n divides m .

Exercise 22.2. In section 22.3.2, we proved Markov's inequality for the special case of $\alpha = 2$. Prove Markov's inequality for general $\alpha > 1$.

Exercise 22.3. Show that the construction given in section 22.3.1 is indeed a universal hash function, using the steps listed below.

To recall the construction, we randomly construct a function $h : [m] \rightarrow [k]$ as follows. First, let p be any prime number $> m$. Draw $a \in \{1, \dots, p-1\}$ uniformly at random, and draw $b \in \{0, \dots, p-1\}$ uniformly at random. We define a function $h(x)$ by

$$h(x) = ((ax + b) \bmod p) \bmod n.$$

1. Let $x_1, x_2 \in [m]$ with $x_1 \neq x_2$, and let $c_1, c_2 \in \{0, \dots, p-1\}$ with $c_1 \neq c_2$. Show that the system of equations

$$\begin{aligned} ax_1 + b &= c_1 \pmod{p} \\ ax_2 + b &= c_2 \pmod{p} \end{aligned}$$

uniquely determines a and b .³

- Step 1 implies that the map $(a, b) \mapsto (ax_1 + b \bmod p, ax_2 + b \bmod p)$ is a bijection.
2. Let $x_1, x_2 \in [m]$ with $x_1 \neq x_2$, and let $c_1, c_2 \in \{0, \dots, p-1\}$ with $c_1 \neq c_2$. Show that

$$\mathbf{P}[ax_1 + b = c_1, ax_2 + b = c_2] = \frac{1}{p(p-1)}.$$

(Here the randomness is over the uniformly random choices of a and b .)

³Here it is helpful to know that division is well-defined in the set of integers $\bmod p$ when p is prime. In particular, " a/b " is defined as the unique integer c such that $bc = a$.

3. Fix $x_1, x_2 \in [m]$ with $x_1 \neq x_2$, and $c_1 \in \{0, \dots, p-1\}$. Show that

$$\sum_{\substack{c_2 \in \{1, \dots, p\} \\ c_2 \neq c_1 \\ c_1 = c_2 \pmod m}} \mathbf{P}[ax + b = c_1, ax + b = c_2] \leq \frac{1}{m}.$$

Hint: You may want to show that the number of values c_2 such that $c_1 = c_2 \pmod m$ is $\leq \frac{p-1}{m}$.

4. Finally, show that $\mathbf{P}[h(x_1) = h(x_2)] \leq \frac{1}{m}$.

Exercise 22.4. The count-min-sketch data structure allows us to estimate the relative frequency of each element up to an ϵ -additive factor with probability of error $\leq 1/\text{poly}(n)$ with $O(\log(n)/\epsilon)$ space. The original motivation, however was to also obtain a list of ϵ -heavy hitters. Explain how to adjust the count-min-sketch to also maintain a collection that contains all of the ϵ -heavy hitters while holding onto at most $O(1/\epsilon)$ items from the stream at any point in time, with probability of error $\leq 1/n^2$. Here your algorithm may also report a limited number of items with relative frequency in the range $[\epsilon/2, \epsilon)$, but no items with relative frequency less than $\epsilon/2$.

Exercise 22.5. In this exercise, we develop a refined analysis that can reduce the additive error substantially in many real settings.

Let S denote the sum of frequency counts of all elements that are *not* ϵ -heavy hitters:

$$S = \sum_{e: p_e < \epsilon} f_e.$$

Note that $S \leq n$, and S might be much less than n when the stream is dominated by heavy hitters.

Show that, by adjusting $\text{count-min-sketch}(\epsilon, 1/n^2)$ slightly by increasing w to $\lceil 4/\epsilon \rceil$, the additive error for every element is at most ϵS with probability of error $\leq 1/n$.