Frequency Estimators

Outline for Today

- ***Randomized Data Structures***
  - Our next approach to improving performance.
- ***Count-Min Sketches***
  - A simple and powerful data structure for estimating frequencies.
- ***Count Sketches***
  - Another approach for estimating frequencies.

Randomized Data Structures

Tradeoffs

- Data structure design is all about tradeoffs:
  - Trade preprocessing time for query time.
  - Trade asymptotic complexity for constant factors.
  - Trade worst-case per-operation guarantees for worst-case aggregate guarantees.

Randomization

- Randomization opens up new routes for tradeoffs in data structures:
  - Trade worst-case guarantees for average-case guarantees.
  - Trade exact answers for approximate answers.
- Over the next few lectures, we'll explore two families of data structures that make these tradeoffs:
  - Today: ***Frequency estimators.***
  - Next Week: ***Hash tables.***

Preliminaries: ***Classes of Hash Functions***

Hashing in Practice

- In most programming languages, each object has “a” hash code.
  - C++: `std::hash`
  - Java: `Object.hashCode`
  - Python: `__hash__`
- Most algorithms and data structures that involve hash functions will not work if objects have just a single hash code.
- Typically, we model hash functions as mathematical functions from a universe $\mathscr{U}$ to some set $\{0, 1, \ldots, m - 1\}$, then consider sets of these functions.
- We can then draw a random function from the set to serve as our hash function.

Universal Hash Functions

- ***Notation:*** Let $[m] = \{0, 1, 2, \ldots, m - 1\}$.
- A set $\mathscr{H}$ is called a ***universal family of hash functions*** if it is a set of functions from $\mathscr{U}$ to $[m]$ where for any distinct $x, y \in \mathscr{U}$, we have

$$\Pr_{h \in \mathscr{H}} [h(x) = h(y)] \leq \frac{1}{m}$$

- Intuitively, universal families of hash functions are classes of hash functions with low collision probabilities.

Pairwise Independence

- A set $\mathscr{H}$ of hash functions from $\mathscr{U}$ to $[m]$ is called ***pairwise independent*** if for any distinct x, $y \in \mathscr{U}$ and for any s, $t \in [m]$, the following holds:
$$\Pr_{h \in \mathscr{H}}[h(x)=s \text{ and } h(y)=t] = \frac{1}{m^2}$$
- Equivalently, $h(x)$ is uniformly distributed over $[m]$. for any $x \in \mathscr{U}$, and for any distinct x and y, the variables $h(x)$ and $h(y)$ are independent.
- If $\mathscr{H}$ is a family of pairwise independent hash functions, then
$$\Pr_{h \in \mathscr{H}}[h(x)=h(y)] = \frac{1}{m}$$

Representing Families

- If any element of $\mathscr{U}$ fits into O(1) machine words, there are pairwise independent families that need O(1) space per function and can be evaluated in time O(1).
- Check CLRS for details.

Preliminaries: ***Vector Norms***

L_1 and L_2 Norms

- Let $\boldsymbol{a} \in \mathbb{R}^n$ be a vector.
- The ***L_1 norm of a***, denoted $\|\boldsymbol{a}\|_1$, is defined as

$$\|\boldsymbol{a}\|_1 = \sum_{i=1}^{n} |\boldsymbol{a}_i|$$

- The ***L_2 norm of a***, denoted $\|\boldsymbol{a}\|_2$, is defined as

$$\|\boldsymbol{a}\|_2 = \sqrt{\sum_{i=1}^{n} \boldsymbol{a}_i^2}$$

Properties of Norms

- The following property of norms holds for any vector $\boldsymbol{a} \in \mathbb{R}^n$. It's a good exercise to prove this on your own:

$$\|\boldsymbol{a}\|_2 \leq \|\boldsymbol{a}\|_1 \leq \Theta(n^{1/2}) \cdot \|\boldsymbol{a}\|_2$$

- The first bound is tight when exactly one component of $\boldsymbol{a}$ is nonzero.
- The second bound is tight when all components of $\boldsymbol{a}$ are equal.

Frequency Estimation

Frequency Estimators

- A ***frequency estimator*** is a data structure supporting the following operations:
  - ***increment***(x), which increments the number of times that x has been seen, and
  - ***estimate***(x), which returns an estimate of the frequency of x.
- Using BSTs, we can solve this in space $\Theta(n)$ with worst-case $O(\log n)$ costs on the operations.
- Using hash tables, we can solve this in space $\Theta(n)$ with expected $O(1)$ costs on the operations.

Frequency Estimators

- Frequency estimation has many applications:
  - Search engines: Finding frequent search queries.
  - Network routing: Finding common source and destination addresses.
- In these applications, $\Theta(n)$ memory can be impractical.
- Unfortunately, this much memory is needed to be able to exactly answer queries.
- ***Goal:*** Get *approximate* answers to these queries in sublinear space.

Some Terminology

- Let's suppose that all elements x are drawn from some set $\mathscr{U} = \{ x_1, x_2, \dots x_n \}$.
- We can interpret the frequency estimation problem as follows:

  Maintain an n-dimensional vector $\boldsymbol{a}$ such that $\boldsymbol{a}_i$ is the frequency of x_i.

- We'll represent $\boldsymbol{a}$ implicitly in a format that uses reduced space.

Where We're Going

- Today, we'll see two data frequency estimation data structures.
- Each is parameterized over two quantities:
  - An ***accuracy*** parameter $\varepsilon \in (0, 1]$ determining how close to accurate we want our answers to be.
  - A ***confidence*** parameter $\delta \in (0, 1]$ determining how likely it is that our estimate is within the bounds given by ε.

Where We're Going

- The ***count-min sketch*** provides estimates with error at most $\varepsilon\|\boldsymbol{a}\|_1$ with probability at least $1 - \delta$.
- The ***count sketch*** provides estimates with an error at most $\varepsilon\|\boldsymbol{a}\|_2$ with probability at least $1 - \delta$.
  - (Notice that lowering ε and lower δ give better bounds.)
- Count-min sketches will use less space than count sketches for the same ε and δ, but provide slightly weaker guarantees.
- Count-min sketches require only universal hash functions, while count sketches require pairwise independence.

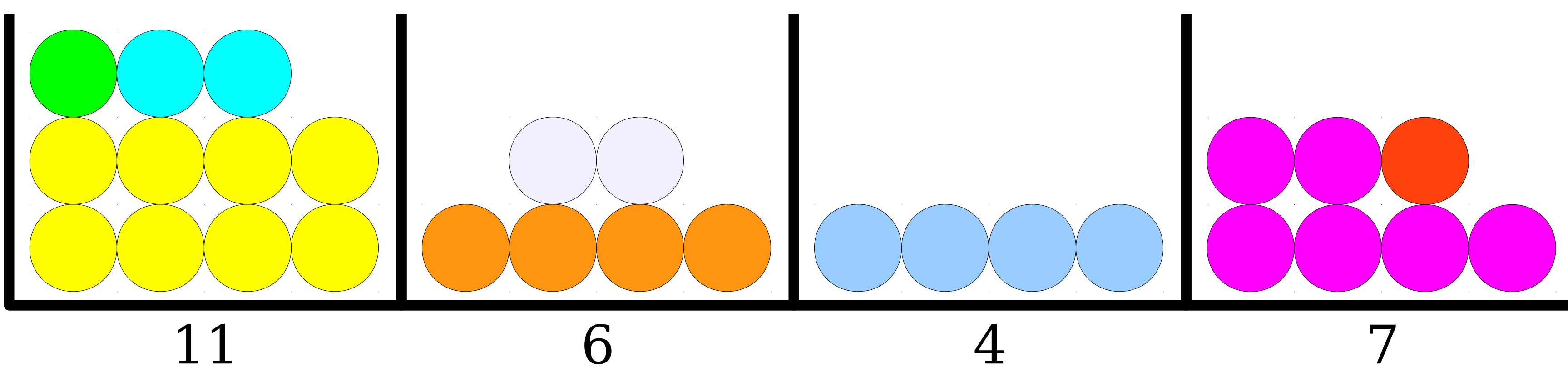
The Count-Min Sketch

The Count-Min Sketch

- Rather than diving into the full count-min sketch, we'll develop the data structure in phases.
- First, we'll build a simple data structure that *on expectation* provides good estimates, but which does not have a high probability of doing so.
- Next, we'll combine several of these data structures together to build a data structure that has a high probability of providing good estimates.

Revisiting the Exact Solution

- In the exact solution to the frequency estimation problem, we maintained a single counter for each distinct element. This is too space-inefficient.
- ***Idea:*** Store a fixed number of counters and assign a counter to each $x_i \in \mathscr{U}$. Multiple x_i's might be assigned to the same counter.
- To ***increment***(x), increment the counter for x.
- To ***estimate***(x), read the value of the counter for x.



Our Initial Structure

- We can formalize this intuition by using universal hash functions.
- Create an array **count** of w counters, each initially zero.
  - We'll choose w later on.
- Choose, from a family $\mathscr{H}$ of universal hash functions, a hash function $h : \mathscr{U} \to [w]$.
- To ***increment***(x), increment **count**$[h(x)]$.
- To ***estimate***(x), return **count**$[h(x)]$.

Analyzing this Structure

- ***Recall:*** $\boldsymbol{a}$ is the vector representing the true frequencies of the elements.
  - $\boldsymbol{a}_i$ is the frequency of element x_i.
- Denote by $\hat{\boldsymbol{a}}_i$ the value of ***estimate***(x_i). This is a random variable that depends on the true frequencies $\boldsymbol{a}$ and the hash function h chosen.
- ***Goal:*** Show that on expectation, $\hat{\boldsymbol{a}}_i$ is not far from $\boldsymbol{a}_i$.

Analyzing this Structure

- Let's look at $\hat{\boldsymbol{a}}_i$ for some choice of x_i.
- The value of $\hat{\boldsymbol{a}}_i$ is given by the true value of $\boldsymbol{a}_i$, plus the frequencies of all of the other elements that hash into the same bucket as x_i.
- To account for the collisions, for each $i \neq j$, introduce a random variable X_j defined as follows:

$$X_j = \begin{cases} \boldsymbol{a}_j & \text{if } h(x_i) = h(x_j) \\ 0 & \text{otherwise} \end{cases}$$

- The value of $\hat{\boldsymbol{a}}_i$ is then given by

$$\hat{\boldsymbol{a}}_i = \boldsymbol{a}_i + \sum_{j \neq i} X_j$$

$$
\begin{aligned}
\mathrm{E}[\hat{\boldsymbol{a}}_i] &= \mathrm{E}[\boldsymbol{a}_i + \sum_{j \neq i} X_j] \\
&= \mathrm{E}[\boldsymbol{a}_i] + \mathrm{E}[\sum_{j \neq i} X_j] \\
&= \boldsymbol{a}_i + \sum_{j \neq i} \mathrm{E}[X_j] \\
&\leq \boldsymbol{a}_i + \sum_{j \neq i} \frac{\boldsymbol{a}_j}{w} \\
&\leq \boldsymbol{a}_i + \frac{\|\boldsymbol{a}\|_1}{w}
\end{aligned}
$$

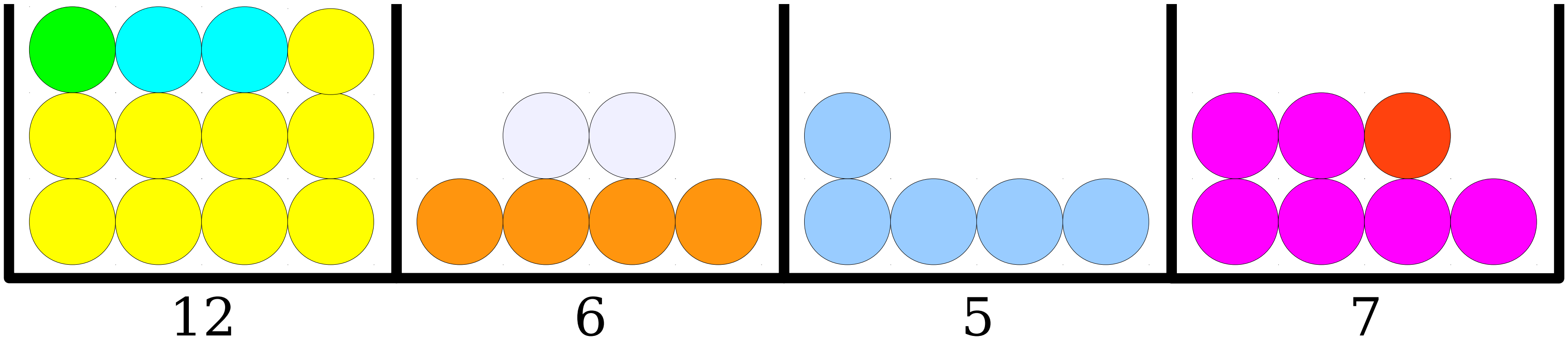
$$
\begin{aligned}
\mathrm{E}[X_j] &= \boldsymbol{a}_j \cdot \Pr[h(x_i) = h(x_j)] + 0 \cdot \Pr[h(x_i) \neq h(x_j)] \\
&= \boldsymbol{a}_j \cdot \Pr[h(x_i) = h(x_j)] \\
&\leq \frac{\boldsymbol{a}_j}{w}
\end{aligned}
$$

Analyzing this Structure

- On expectation, the value of ***estimate***(x_i) is at most $\|\boldsymbol{a}\|_1 / w$ greater than a_i.
- Intuitively, this makes sense; this is what you'd get if all the extra error terms were distributed uniformly across the counters.
- Increasing w increases memory usage, but improves accuracy.
- Decreasing w decreases memory usage, but decreases accuracy.

One Problem

- We have shown that *on expectation*, the value of ***estimate***(x_i) can be made close to the true value.
- However, this data structure may give wildly inaccurate results for most elements.
  - Any low-frequency elements that collide with high-frequency elements will have overreported frequency.



One Problem

- We have shown that *on expectation*, the value of ***estimate***(x_i) can be made close to the true value.
- However, this data structure may give wildly inaccurate results for most elements.
  - Any low-frequency elements that collide with high-frequency elements will have overreported frequency.
- ***Question:*** Can we bound the probability that we overestimate the frequency of an element?

A Useful Observation

- Notice that regardless of which hash function we use or the size of the table, we always have $\hat{a}_i \geq a_i$.
- This means that $\hat{a}_i - a_i \geq 0$.
- We have a ***one-sided error***; this data structure will never underreport the frequency of an element, but it may overreport it.

Bounding the Error Probability

- If X is a nonnegative random variable, then ***Markov's inequality*** states that for any $c > 0$, we have

$$\Pr[X > c \cdot \mathrm{E}[X]] \leq 1/c$$

- We know that

$$\mathrm{E}[\hat{\boldsymbol{a}}_i] \leq \boldsymbol{a}_i + \|\boldsymbol{a}\|_1 / w$$

- Therefore, we see

$$\mathrm{E}[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i] \leq \|\boldsymbol{a}\|_1 / w$$

- By Markov's inequality, for any $c > 0$, we have

$$\Pr\left[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i > \frac{c\|\boldsymbol{a}\|_1}{w}\right] \leq 1/c$$

- Equivalently:

$$\Pr\left[\hat{\boldsymbol{a}}_i > \boldsymbol{a}_i + \frac{c\|\boldsymbol{a}\|_1}{w}\right] \leq 1/c$$

Bounding the Error Probability

- For any $c > 0$, we know that

$$\Pr[\hat{\boldsymbol{a}}_i > \boldsymbol{a}_i + \frac{c\|\boldsymbol{a}\|_1}{w}] \leq 1/c$$

- In particular:

$$\Pr[\hat{\boldsymbol{a}}_i > \boldsymbol{a}_i + \frac{e\|\boldsymbol{a}\|_1}{w}] \leq 1/e$$

- Given any $0 < \varepsilon < 1$, let's set $w = \lceil e / \varepsilon \rceil$. Then we have

$$\Pr[\hat{\boldsymbol{a}}_i > \boldsymbol{a}_i + \varepsilon\|\boldsymbol{a}\|_1] \leq 1/e$$

- This data structure uses $O(\varepsilon^{-1})$ space and gives estimates with error at most $\varepsilon\|\boldsymbol{a}\|_1$ with probability at least $1 - 1 / e$.

Tuning the Probability

- Right now, we can tune the accuracy ε of the data structure, but we can't tune our *confidence* in that answer (it's always 1 - 1 / *e*).
- ***Goal:*** Update the data structure so that for any confidence $0 < \delta < 1$, the probability that an estimate is correct is at least $1 - \delta$.

Tuning the Probability

- If this structure has a constant probability of giving a good estimate, many copies of this structure in parallel have an even better chance.
- ***Idea:*** Combine together multiple copies of this data structure to boost confidence in our estimates.

Running in Parallel

- Let's suppose that we run d independent copies of this data structure. Each has its own independently randomly chosen hash function.
- To ***increment***(x) in the overall structure, we call ***increment***(x) on each of the underlying data structures.
- The probability that at least one of them provides a good estimate is quite high.
- ***Question:*** How do you know which one?

Recognizing the Answer

- ***Recall:*** Each estimate $\hat{\boldsymbol{a}}_i$ is the sum of two independent terms:
  - The actual value $\boldsymbol{a}_i$.
  - Some "noise" terms from other elements colliding with x_i.
- Since the noise terms are always nonnegative, larger values of $\hat{\boldsymbol{a}}_i$ are less accurate than smaller values of $\hat{\boldsymbol{a}}_i$.
- ***Idea:*** Take, as our estimate, the minimum value of $\hat{\boldsymbol{a}}_i$ from all of the data structures.

The Final Analysis

- For each independent copy of this data structure, the probability that our estimate is within $\varepsilon||\boldsymbol{a}||_1$ of the true value is at least $1 - 1 / e$.
- Let $\mathcal{E}_i$ be the event that the ith copy of the data structure provides an estimate within $\varepsilon||\boldsymbol{a}||_1$ of the true answer.
- Let $\mathcal{E}$ be the event that the aggregate data structure provides an estimate within $\varepsilon||\boldsymbol{a}||_1$.
- ***Question:*** What is $\Pr[\mathcal{E}]$?

The Final Analysis

- Since we're taking the minimum of all the estimates, if *any* of the data structures provides a good estimate, our estimate will be accurate.
- Therefore,

$$\Pr[\mathcal{E}] = \Pr[\exists i.\ \mathcal{E}_i]$$

- Equivalently:

$$\Pr[\mathcal{E}] = 1 - \Pr[\forall i.\ \overline{\mathcal{E}}_i]$$

- Since all the estimates are independent:

$$\Pr[\mathcal{E}] = 1 - \Pr[\forall i.\ \overline{\mathcal{E}}_i] \geq 1 - 1/e^d.$$

The Final Analysis

- We now have that

$$\Pr[\mathcal{E}] \leq 1 - 1/e^{d}.$$

- If we want the confidence to be $1 - \delta$, we can choose δ such that

$$1 - \delta = 1 - 1/e^{d}$$

- Solving, we can choose $d = \ln \delta^{-1}$.
- If we make $\ln \delta^{-1}$ independent copies of our data structure, the probability that our estimate is off by at most $\varepsilon||\boldsymbol{a}||_1$ is at least $1 - \delta$.

The Count-Min Sketch

- This data structure is called a ***count-min sketch***.
- Given parameters ε and δ, choose

$$w = \lceil e \,/\, \varepsilon \rceil \qquad d = \lceil \ln \delta^{-1} \rceil$$

- Create an array **`count`** of size $w \times d$ and for each row i, choose a hash function $h_i : \mathscr{U} \to [w]$ independently from a universal family of hash functions $\mathscr{H}$.
- To ***increment***(x), increment **`count`**$[i][h_i(x)]$ for each row i.
- To ***estimate***(x), return the minimum value of **`count`**$[i][h_i(x)]$ across all rows i.

The Count-Min Sketch

- Update and query times are $O(d)$, which is $O(\log \delta^{-1})$.
- Space usage: $O(\varepsilon^{-1} \cdot \log \delta^{-1})$ counters.
  - This can be *significantly* better than just storing a raw frequency count!
- Provides an estimate to within $\varepsilon \|\boldsymbol{a}\|_1$ with probability at least $1 - \delta$.

The General Pattern

- At a high level, the data structure works as follows:
  - Create an array of counters tracking the frequencies of various elements.
  - Bound the probability that the estimate deviates significantly from the true value.
  - Store multiple independent copies of this data structure.
  - Find a way to aggregate information across the copies.
  - Bound the probability that the aggregate is wrong across all instances.
- This same intuition forms the basis for the count sketch, which we'll see next.

Time-Out for Announcements!

Final Project Proposal

- As a reminder, final project proposals are due this upcoming Tuesday, May 10.
- We sent out a link earlier today you can use to submit your ranked project topics and your team members.
- Don't have a team yet? Stick around after class for a meet-and-mingle, or use the find-a-partner feature on Piazza!

Problem Set Three

- PS3 has been graded and grades are now available on GradeScope.

Back to CS166!

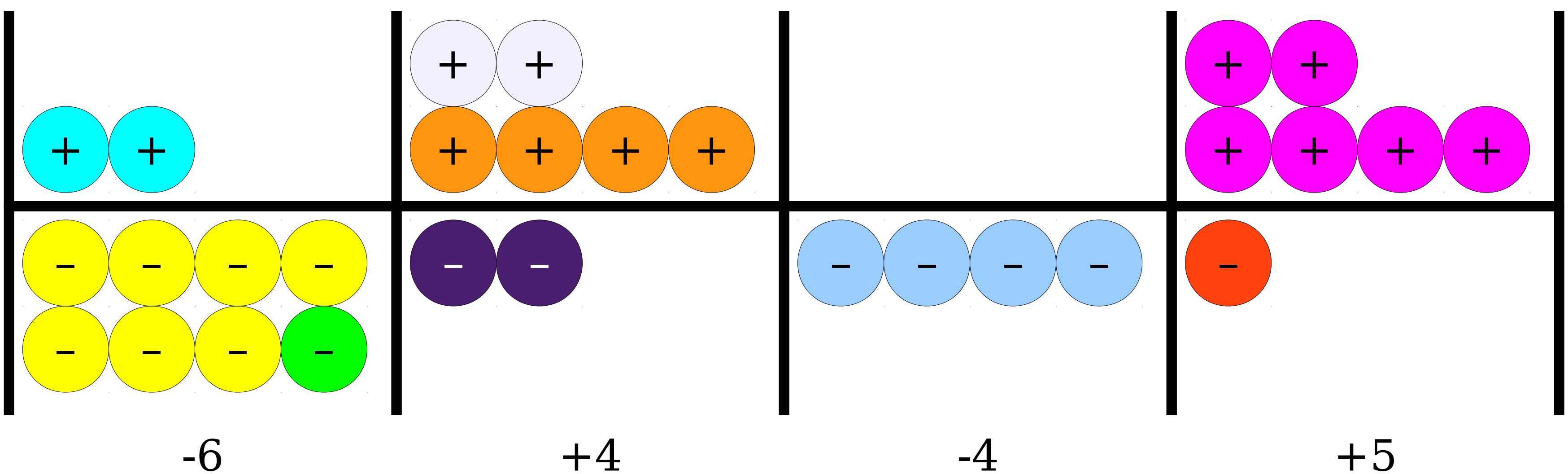
An Alternative: Count Sketches

The Motivation

- *(Note: This is historically backwards; count sketches came before count-min sketches.)*
- In a count-min sketch, errors arise when multiple elements collide.
- Errors are strictly additive; the more elements collide in a bucket, the worse the estimate for those elements.
- ***Question:*** Can we try to offset the “badness” that results from the collisions?

The Setup

- As before, for some parameter w, we'll create an array **`count`** of length w.
- As before, choose a hash function $h : \mathscr{U} \to [w]$ from a family $\mathscr{H}$.
- For each $x_i \in \mathscr{U}$, assign x_i either +1 or -1.
- To ***increment***(x), go to **`count`**$[h(x)]$ and add ± 1 as appropriate.
- To ***estimate***(x), return **`count`**$[h(x)]$, multiplied by ± 1 as appropriate.



The Intuition

- Think about what introducing the ± 1 term does when collisions occur.
- If an element x collides with a frequent element y, we're not going to get a good estimate for x (but we wouldn't have gotten one anyway).
- If x collides with multiple infrequent elements, the collisions between those elements will partially offset one another and leave a better estimate for x.

More Formally

- Let's formalize this idea more concretely.
- In addition to choosing $h \in \mathscr{H}$, choose a second hash function $s : \mathscr{U} \to \{+1, -1\}$ from a pairwise independent family $\mathscr{S}$.
- ***Assumption:*** The functions in $\mathscr{H}$ are independent of the functions in $\mathscr{S}$.
- To ***increment***(x), add $s(x)$ to **`count`**$[h(x)]$.
- To ***estimate***(x), return $s(x) \cdot$ **`count`**$[h(x)]$.

How accurate is our estimation?

Formalizing the Intuition

- As before, define $\hat{\boldsymbol{a}}_i$ to be our estimate of $\boldsymbol{a}_i$.
- As before, $\hat{\boldsymbol{a}}_i$ will depend on how the other elements are distributed. Unlike before, it now also depends on signs given to the elements by s.
- Specifically, for each other x_j that collides with x_i, the error contribution will be

$$s(x_i) \cdot s(x_j) \cdot \boldsymbol{a}_j$$

- Why?
  - The counter for x_i will have $s(x_j)\ \boldsymbol{a}_j$ added in.
  - We multiply the counter by $s(x_i)$ before returning it.

Properties of s

Claim: $\Pr[s(x_i) \cdot s(x_j) = 1] = ½$.

Proof: The product is 1 iff both of the signs are +1 or both the signs are -1.

Using the definition of a pairwise independent family of hash functions, each of those possibilities has probability ¼ of occurring.

Since they're mutually exclusive, the overall probability is ½. ■

Formalizing the Intuition

- As with count-min sketches, we'll introduce random variables representing the error contribution from other elements.
- For all $j \neq i$, let X_j be a random variable defined as follows:

$$X_j = \begin{cases} a_j & \text{if } h(i) = h(j) \text{ and } s(x_i)s(x_j) = 1 \\ 0 & \text{if } h(i) \neq h(j) \\ -a_j & \text{if } h(i) = h(j) \text{ and } s(x_i)s(x_j) = -1 \end{cases}$$

- Notice that $\mathrm{E}[X_j] = 0$.

Formalizing the Intuition

- We can now express $\boldsymbol{\hat{a}}_i$ in terms of these X_j variables.
- Specifically, $\boldsymbol{\hat{a}}_i$ is given by the true value of $\boldsymbol{a}_i$ plus extra amounts from collisions.
- Mathematically:

$$\boldsymbol{\hat{a}}_i = \boldsymbol{a}_i + \sum_{j \neq i} X_j$$

Computing the Expectation

- Something interesting happens when we compute $\mathrm{E}[\hat{\boldsymbol{a}}_i]$:

$$
\begin{aligned}
\mathrm{E}[\hat{\boldsymbol{a}}_i] &= \mathrm{E}[\boldsymbol{a}_i + \sum_{j \neq i} X_j] \\
&= \mathrm{E}[\boldsymbol{a}_i] + \mathrm{E}[\sum_{j \neq i} X_j] \\
&= \boldsymbol{a}_i + \sum_{j \neq i} \mathrm{E}[X_j] \\
&= \boldsymbol{a}_i
\end{aligned}
$$

- On expectation, we get the exact value of $\boldsymbol{a}_i$!
- How likely is this?

A Hitch

- In the count-min sketch, we used Markov's inequality to bound the probability that we get a bad estimate.
- This worked because $\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i$ was a nonnegative random variable.
- However, $\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i$ can be negative in the count sketch because collisions can decrease the estimate $\hat{\boldsymbol{a}}_i$ below the true value $\boldsymbol{a}_i$.
- We'll need to use a different technique to bound the error.

Chebyshev to the Rescue

- ***Chebyshev's inequality*** states that for any random variable X with finite variance, given any $c > 0$, the following holds:

$$\Pr\left[\, |X - \mathrm{E}[X]| \geq c\sqrt{\mathrm{Var}[X]} \,\right] \leq \frac{1}{c^2}$$

- If we can get the variance of $\hat{\boldsymbol{a}}_i$, we can bound the probability that we get a bad estimate with our data structure.

Computing the Variance

- Let's go compute Var[$\hat{\boldsymbol{a}}_i$]:

$$\begin{aligned} \mathrm{Var}[\hat{\boldsymbol{a}}_i] &= \mathrm{Var}[\boldsymbol{a}_i + \sum_{j \neq i} X_j] \\ &= \mathrm{Var}[\sum_{j \neq i} X_j] \\ &= \sum_{j \neq i} \mathrm{Var}[X_j] \end{aligned}$$

- Although Var is not a linear operator, because our hash function is pairwise independent, all of the X_j's are pairwise independent.
- Therefore, the variance of the sum is the sum of the variances.

Computing the Variance

- ***Recall:*** Var[X_j] = E[X_j^2] – E[X_j]^2.
- We know that for all X_j that E[X_j] = 0.
- We can determine E[X_j^2] by looking at X_j^2:

$$X_j = \begin{cases} a_j & \text{if } h(i) = h(j) \text{ and } s(x_i)s(x_j) = 1 \\ 0 & \text{if } h(i) \neq h(j) \\ -a_j & \text{if } h(i) = h(j) \text{ and } s(x_i)s(x_j) = -1 \end{cases}$$

$$X_j^2 = \begin{cases} a_j^2 & \text{if } h(i) = h(j) \\ 0 & \text{if } h(i) \neq h(j) \end{cases}$$

- Therefore, E[X_j^2] = a_j^2 Pr[$h(i) = h(j)$] = $\boldsymbol{a_j^2 / w}$.

Using the Variance

$$
\begin{aligned}
\mathrm{Var}[\hat{\boldsymbol{a}}_i] &= \mathrm{Var}[\boldsymbol{a}_i + \sum_{j \neq i} X_j] \\
&= \mathrm{Var}[\sum_{j \neq i} X_j] \\
&= \sum_{j \neq i} \mathrm{Var}[X_j] \\
&= \sum_{j \neq i} \frac{\boldsymbol{a}_j^2}{w} \\
&\leq \frac{\|\boldsymbol{a}\|_2^2}{w}
\end{aligned}
$$

Harnessing Chebyshev

- Chebyshev's Inequality says

$$\Pr\left[\ \left|X - \mathrm{E}[X]\right| \geq c\sqrt{\mathrm{Var}[X]}\ \right] \leq 1/c^2$$

- Applying this to $\hat{\boldsymbol{a}}_i$ yields

$$\Pr\left[\ \left|\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i\right| \geq \frac{c\left\|\boldsymbol{a}\right\|_2}{\sqrt{w}}\ \right] \leq 1/c^2$$

- For any ε, choose $w = \lceil e / \varepsilon^2 \rceil$, so

$$\Pr\left[\ \left|\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i\right| \geq \frac{c\,\varepsilon\left\|\boldsymbol{a}\right\|_2}{\sqrt{e}}\ \right] \leq 1/c^2$$

- Therefore, choosing $c = e^{1/2}$ gives

$$\Pr\left[\ \left|\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i\right| \geq \varepsilon\left\|\boldsymbol{a}\right\|_2\ \right] \leq 1/e$$

The Story So Far

- We now know that, by setting $\varepsilon = (e / w)^{1/2}$, the estimate is within $\varepsilon\|\boldsymbol{a}\|_2$ with probability at least $1 - 1 / e$.
- Solving for w, this means that we will choose $w = \lceil e / \varepsilon^2 \rceil$.
- Space usage is now $O(\varepsilon^{-2})$, but the error bound is now $\varepsilon\|\boldsymbol{a}\|_2$ rather than $\varepsilon\|\boldsymbol{a}\|_1$.
- As before, the next step is to reduce the error probability.

Repetitions with a Catch

- As before, our goal is to make it possible to choose a bound $0 < \delta < 1$ so that the confidence is at least $1 - \delta$.
- As before, we'll do this by making *d* independent copies of the data structure and running each in parallel.
- Unlike the count-min sketch, errors in count sketches are two-sided; we can overshoot or undershoot.
- Therefore, it's not meaningful to take the minimum or maximum value.
- How do we know which value to report?

Working with the Median

- ***Claim:*** If we output the median estimate given by the data structures, we have high probability of giving the right answer.
- ***Intuition:*** The only way we report an answer more than $\varepsilon||\boldsymbol{a}||_2$ is if at least half of the data structures output an answer that is more than $\varepsilon||\boldsymbol{a}||_2$ from the true answer.
- Each individual data structure is wrong with probability at most 1 / *e*, so this is highly unlikely.

The Setup

- Let X denote a random variable equal to the number of data structures that produce an answer *not* within $\varepsilon\|\boldsymbol{a}\|_2$ of the true answer.
- Since each independent data structure has failure probability at most $1 / e$, we can upper-bound X with a $\text{Binom}(d, 1 / e)$ variable.
- We want to know $\Pr[X > d / 2]$.
- How can we determine this?

Chernoff Bounds

- The ***Chernoff bound*** says that if $X \sim \text{Binom}(n, p)$ and $p < 1/2$, then

$$\Pr[X > n/2] < e^{\frac{-n(1/2-p)^2}{2p}}$$

- In our case, $X \sim \text{Binom}(d, 1/e)$, so we know that

$$\begin{aligned}\Pr\left[X > \frac{d}{2}\right] &\leq e^{\frac{-d(1/2-1/e)^2}{2(1/e)}} \\ &= e^{-\mathrm{O}(1)\cdot d}\end{aligned}$$

- Therefore, choosing $d = \mathrm{O}(\log \delta^{-1})$ ensures that $\Pr[X > d / 2] \leq \delta$.
- Therefore, the success probability is at least $1 - \delta$.

Chernoff Bounds

- The ***Chernoff bound*** says that if $X \sim \text{Binom}(n, p)$ and $p < 1/2$, then

$$\Pr[X > n/2] < e^{\frac{-n(1/2-p)^2}{2p}}$$

- d, $1/e$), so we know that

Notice that the constant factor hidden here is significant, since it's used in an exponent. If you want to use this in practice, you'll have to work out the math to determine the optimal constant to use.

$$e^{\frac{-d(1/2-1/e)^2}{2(1/e)}}$$

$$e^{-O(1)\cdot d}$$

- Therefore, choosing $d = O(\log \delta^{-1})$ ensures that $\Pr[X > d / 2] \leq \delta$.
- Therefore, the success probability is at least $1 - \delta$.

The Overall Construction

- The ***count sketch*** is the data structure given as follows.
- Given ε and δ, choose

$$w = \lceil e / \varepsilon^2 \rceil \qquad d = O(\log \delta^{-1})$$

- Create an array **`count`** of $w \times d$ counters.
- Choose hash functions h_i and s_i for each of the d rows.
- To ***increment***(x), add $s_i(x)$ to **`count`**$[i][h_i(x)]$ for each row i.
- To ***estimate***(x), return the median of $s_i(x) \cdot$ **`count`**$[i][h_i(x)]$ for each row i.

The Final Analysis

- With probability at least $1 - \delta$, all estimates are accurate to within a factor of $\varepsilon \lVert \boldsymbol{a} \rVert_2$.
- Space usage is $O(w \times d)$, which we've seen to be $O(\varepsilon^{-2} \cdot \log \delta^{-1})$.
- Updates and queries run in time $O(\delta^{-1})$.
- Trades factor of ε^{-1} space for an accuracy guarantee relative to $\lVert \boldsymbol{a} \rVert_2$ versus $\lVert \boldsymbol{a} \rVert_1$.

In Practice

- These data structures have been and continue to be used in practice.
- These sketches and their variants have been used at Google and Yahoo! (or at least, there are papers coming from there about their usage).
- Many other sketches exist as well for estimating other quantities; they'd make for really interesting final project topics!

More to Explore

- A ***cardinality estimator*** is a data structure for estimating how many different elements have been seen in sublinear time and space. They're used extensively in database implementations.
- If instead of estimating $\boldsymbol{a}_i$ terms individually we want to estimate $\|\boldsymbol{a}\|_1$ or $\|\boldsymbol{a}_2\|$, we can use a ***frequency moment estimator***.
  - You'll see one of them, the *tug-of-war sketch*, on Problem Set Five.
- These would make for really interesting final project topics!

Some Concluding Notes

Randomized Data Structures

- You may have noticed that the final versions of these data structures are actually not all that complex – each just maintains a set of hash functions and some 2D tables.
- The analyses, on the other hand, are a lot more involved than what we saw for other data structures.
- This is common – randomized data structures often have simple descriptions and quite complex analyses.

The Strategy

- Typically, an analysis of a randomized data structure looks like this:
  - First, show that the data structure (or some random variable related to it), on expectation, performs well.
  - Second, use concentration inequalities (Markov, Chebyshev, Chernoff, or something else) to show that it's unlikely to deviate from expectation.
- The analysis often relies on properties of some underlying hash function. On Tuesday, we'll explore why this is so important.

Next Time

- ***Hashing Strategies***
  - There are a lot of hash tables out there. What do they look like?
- ***Linear Probing***
  - The original hashing strategy!
- ***Analyzing Linear Probing***
  - ...is way, way more complicated than you probably would have thought. But it's beautiful! And a great way to learn about randomized data structures!