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: *What is a Hash Function?*
Hashing in Practice

- In most programming languages, each object has “a” hash code.
  - C++: `std::hash`
  - Java: `Object.hashCode`
  - Python: `__hash__`
- To store objects in a hash table, you just go and implement the appropriate function or type.
- In other words, hash functions are *intrinsic* properties of objects.
Hashing in Theoryland

• In Theoryland, a hash function is a function from some domain called the **universe** (typically denoted $\mathcal{U}$) to some codomain.

• The codomain is usually a set of the form $\{0, 1, 2, 3, \ldots, m - 1\}$, which we’ll denote $[m]$.

• We often will grab lots of different hash functions from the same universe $\mathcal{U}$ to some codomain, and we’ll assume we have access to as many of them as we need.

• In other words, hash functions are **extrinsic** to objects, and it’s possible to have multiple different hash functions available at the same time.
Families of Hash Functions

• A *family* of hash functions is a set $\mathcal{H}$ of hash functions with the same domain and codomain.

• The data structures we’ll explore will assume that we have access to certain families of hash functions with nice properties.

• We’ll then sample uniformly-random choices $h \in \mathcal{H}$ to use as needed.
Sampling Random Functions

- Here’s a family of hash functions \( \mathcal{H} \) from \( \mathbb{N} \) to [137]:
  \[
  \mathcal{H} = \{ f(n) = (an + b) \mod 137 \mid a, b \in [137] \}
  \]
- In Theoryland, we’d model picking a uniformly-random hash function from \( \mathcal{H} \) as just that – sampling some \( h \in \mathcal{H} \) uniformly.
- In The Real World, we’d probably model picking such a function like this:
  ```c
  int a = rand() % 137;
  int b = rand() % 137;

  int hash(int value) {
    return (a * value + b) % 137;
  }
  ```
Characterizing Hash Functions

- Different algorithms and data structures require different guarantees from their hash functions.

- In CS161, you explored universal hash functions in the context of chained hash tables.

- For what we’ll be doing in CS166, we’re going to need hash functions with slightly stronger probabilistic guarantees.
Pairwise Independence

- Let $\mathcal{H}$ be a family of hash functions from $\mathcal{U}$ to some set $\mathcal{C}$.
- We say that $\mathcal{H}$ is a **2-independent family of hash functions** if, for any distinct distinct $x, y \in \mathcal{U}$, if we choose a hash function $h \in \mathcal{H}$ uniformly at random, the following hold:

\[
\begin{align*}
h(x) & \text{ and } h(y) \text{ are uniformly distributed over } \mathcal{C}. \\
h(x) \text{ and } h(y) \text{ are independent.}
\end{align*}
\]

- 2-independent hash functions are great hash functions when we want a nice distribution over the output space even after fixing some specific element.
3-Independence

- Let $\mathcal{H}$ be a family of hash functions from $\mathcal{U}$ to some set $\mathcal{C}$.
- We say that $\mathcal{H}$ is a 3-independent family of hash functions if, for any distinct distinct $x, y, z \in \mathcal{U}$, if we choose a hash function $h \in \mathcal{H}$ uniformly at random, the following hold:
  
  $h(x), h(y), \text{ and } h(z) \text{ are uniformly distributed over } \mathcal{C}$.
  
  $h(x), h(y), \text{ and } h(z) \text{ are independent}$.

- As you’ll see, in many cases, making stronger assumptions about our hash functions makes it possible to simplify tricky probabilistic expressions.

- (As you can probably guess, this generalizes even further to $k$-independence, which we’ll see on Tuesday.)
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.
- **Goal**: Get *approximate* answers to these queries in sublinear space.
Some Terminology

● Let's suppose that all elements $x$ are drawn from some set $\mathcal{U} = \{ x_1, x_2, \ldots x_n \}$.

● We can interpret the frequency estimation problem as follows:

  Maintain an $n$-dimensional vector $a$ such that $a_i$ is the frequency of $x_i$.

● We'll represent $a$ implicitly in a format that uses reduced space.
Vector Norms

- Let \(a \in \mathbb{R}^n\) be a vector.

- The **L\(_1\) norm of \(a\)**, denoted \(\|a\|_1\), is defined as

\[
\|a\|_1 = \sum_{i=1}^{n} |a_i|
\]

- The **L\(_2\) norm of \(a\)**, denoted \(\|a\|_2\), is defined as

\[
\|a\|_2 = \sqrt{\sum_{i=1}^{n} a_i^2}
\]
Properties of Norms

- The following property of norms holds for any vector \( a \in \mathbb{R}^n \). It's a good exercise to prove this on your own:
  \[
  \|a\|_2 \leq \|a\|_1 \leq \Theta(n^{1/2}) \cdot \|a\|_2
  \]
- The first bound is tight when exactly one component of \( a \) is nonzero.
- The second bound is tight when all components of \( a \) are equal.
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 $\varepsilon$. 
Where We're Going

- The **count-min sketch** provides estimates with error at most $\varepsilon \|a\|_1$ with probability at least $1 - \delta$.
- The **count sketch** provides estimates with an error at most $\varepsilon \|a\|_2$ with probability at least $1 - \delta$.
  - (Notice that lowering $\varepsilon$ and lower $\delta$ give better bounds.)
- Count-min sketches will use less space than count sketches for the same $\varepsilon$ and $\delta$, but provide slightly weaker guarantees.
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 \( x_i \in \mathcal{U} \). 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 model “assigning each $x_i$ to a counter” by using hash functions.
- Choose, from a family of 2-independent hash functions $\mathcal{H}$, a uniformly-random hash function $h : \mathcal{U} \rightarrow [w]$.
- Create an array $\text{count}$ of $w$ counters, each initially zero.
  - We'll choose $w$ later on.
- To $\text{increment}(x)$, increment $\text{count}[h(x)]$.
- To $\text{estimate}(x)$, return $\text{count}[h(x)]$. 
Analyzing this Structure

- **Recall:** $a$ is the vector representing the true frequencies of the elements.
  - $a_i$ is the frequency of element $x_i$.
- Denote by $\hat{a}_i$ the value of $\text{estimate}(x_i)$. This is a random variable that depends on the true frequencies $a$ (out of our control, but not random) and the hash function $h$ (truly chosen at random.)
- **Goal:** Show that on expectation, $\hat{a}_i$ is not far from $a_i$. 
Analyzing this Structure

• Intuitively, what do we expect $\hat{a}_i$ to be?
• There are $\| a \|_1$ total elements spread out across $w$ buckets.
• Assuming they’re well-distributed, we’d probably expect $\| a \|_1 / w$ of them to be in each bucket.
• So a reasonable guess would be that $\hat{a}_i$ should probably end up being something like $a_i + \| a \|_1 / w$.
• Let’s see if we can formalize this.
Analyzing this Structure

• Let's look at $\hat{a}_i = \text{count}[h(x_i)]$ for some choice of $x_i$.
• For each element $x_j$:
  • If $h(x_i) = h(x_j)$, then $x_j$ contributes $a_j$ to $\text{count}[h(x_i)]$.
  • If $h(x_i) \neq h(x_j)$, then $x_j$ contributes 0 to $\text{count}[h(x_i)]$.
• To pin this down precisely, let’s define a set of random variables $X_1, X_2, \ldots$, as follows:
  $$X_j = \begin{cases} 
  1 & \text{if } h(x_i) = h(x_j) \\
  0 & \text{otherwise}
  \end{cases}$$

Each of these variables is called an indicator random variable, since it “indicates” whether some event occurs.
Analyzing this Structure

• Let's look at $\hat{a}_i = \text{count}[h(x_i)]$ for some choice of $x_i$.

• For each element $x_j$:
  • If $h(x_i) = h(x_j)$, then $x_j$ contributes $a_j$ to $\text{count}[h(x_i)]$.
  • If $h(x_i) \neq h(x_j)$, then $x_j$ contributes 0 to $\text{count}[h(x_i)]$.

• To pin this down precisely, let’s define a set of random variables $X_1, X_2, \ldots$, as follows:

\[
X_j = \begin{cases} 
1 & \text{if } h(x_i) = h(x_j) \\
0 & \text{otherwise}
\end{cases}
\]

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

\[
\hat{a}_i = \sum_j a_j X_j = a_i + \sum_{j \neq i} a_j X_j
\]
\[ E[\hat{a}_i] = E[a_i + \sum_{j \neq i} a_j X_j] \]
\[ = E[a_i] + E[\sum_{j \neq i} a_j X_j] \]

This follows from **linearity of expectation**. We’ll use this property extensively over the next few days.
$$E[\hat{a}_i] = E[a_i + \sum_{j \neq i} a_j X_j]$$

$$= E[a_i] + E[\sum_{j \neq i} a_j X_j]$$

$$= a_i + \sum_{j \neq i} E[a_j X_j]$$

The actual value of $a_i$ is not a random variable. The randomness here is in our choice of hash function, not the choice of the data.
\[
E[\hat{a}_i] = E[a_i + \sum_{j \neq i} a_j X_j] \\
= E[a_i] + E[\sum_{j \neq i} a_j X_j] \\
= a_i + \sum_{j \neq i} E[a_j X_j] \\
= a_i + \sum_{j \neq i} a_j E[X_j]
\]

\[
E[X_j] = 1 \cdot \Pr[h(x_i) = h(x_j)] + 0 \cdot \Pr[h(x_i) \neq h(x_j)]
\]

\[
X_j = \begin{cases} 
1 & \text{if } h(x_i) = h(x_j) \\
0 & \text{otherwise}
\end{cases}
\]
$$E[\hat{a}_i] = E[a_i + \sum_{j \neq i} a_j X_j]$$

$$= E[a_i] + E[\sum_{j \neq i} a_j X_j]$$

$$= a_i + \sum_{j \neq i} E[a_j X_j]$$

$$= a_i + \sum_{j \neq i} a_j E[X_j]$$

$$E[X_j] = 1 \cdot \Pr[h(x_i) = h(x_j)] + 0 \cdot \Pr[h(x_i) \neq h(x_j)]$$

$$= 1 \cdot \Pr[h(x_i) = h(x_j)]$$

If $X$ is an indicator variable for some event $\mathcal{E}$, then $E[X] = \Pr[\mathcal{E}]$. This is really useful when using linearity of expectation!
\[ E[\hat{a}_i] = E[a_i + \sum_{j \neq i} a_j X_j] \]
\[ = E[a_i] + E[\sum_{j \neq i} a_j X_j] \]
\[ = a_i + \sum_{j \neq i} E[a_j X_j] \]
\[ = a_i + \sum_{j \neq i} a_j E[X_j] \]

\[ E[X_j] = 1 \cdot \Pr[h(x_i) = h(x_j)] + 0 \cdot \Pr[h(x_i) \neq h(x_j)] \]
\[ = 1 \cdot \Pr[h(x_i) = h(x_j)] \]
\[ = \frac{1}{w} \]

Any two hash codes from a randomly-chosen 2-independent hash function are independent, uniformly-random variables.
\[
E[\hat{a}_i] = E[a_i + \sum_{j \neq i} a_j X_j] \\
= E[a_i] + E[\sum_{j \neq i} a_j X_j] \\
= a_i + \sum_{j \neq i} E[a_j X_j] \\
= a_i + \sum_{j \neq i} a_j E[X_j] \\
= a_i + \sum_{j \neq i} \frac{a_j}{w} \\
\leq a_i + \frac{\|a\|_1}{w}
\]

\[
E[X_j] = 1 \cdot \Pr[h(x_i) = h(x_j)] + 0 \cdot \Pr[h(x_i) \neq h(x_j)] \\
= 1 \cdot \Pr[h(x_i) = h(x_j)] \\
= \frac{1}{w}
\]
Interpreting our Analysis

• On expectation, the value of \( \text{estimate}(x_i) \) is at most \( \|a\|_1 / w \) greater than \( a_i \).
  
  • That matches our intuition from before! Yay!

• From a practical perspective:
  
  • 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 \textit{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 \( \text{estimate}(x_i) \) 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
  \[ \Pr[X > c \cdot E[X]] \leq 1/c \]
- We know that
  \[ E[\hat{a}_i] \leq a_i + \|a\|_1/w \]
- Therefore, we see that
  \[ E[\hat{a}_i - a_i] \leq \|a\|_1/w \]
- By Markov's inequality, for any $c > 0$, we have
  \[ \Pr[\hat{a}_i - a_i > \frac{c \|a\|_1}{w}] \leq 1/c \]
- Equivalently:
  \[ \Pr[\hat{a}_i > a_i + \frac{c \|a\|_1}{w}] \leq 1/c \]
Bounding the Error Probability

• For any $c > 0$, we know that
  \[ \Pr[\hat{a}_i > a_i + \frac{c \|a\|_1}{w}] \leq \frac{1}{c} \]

• In particular:
  \[ \Pr[\hat{a}_i > a_i + \frac{e \|a\|_1}{w}] \leq \frac{1}{e} \]

• Given an accuracy parameter $\varepsilon$, $\in (0, 1]$, let's set $w = \lceil e / \varepsilon \rceil$. Then we have
  \[ \Pr[\hat{a}_i > a_i + \varepsilon \|a\|_1] \leq \frac{1}{e} \]

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

• Right now, we can tune the accuracy $\varepsilon$ 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

- A single copy of our data structure has a decently good chance of providing an estimate that isn’t too far off the true value.

- Intuitively, having *lots* of copies of this data structure would make it more likely that at least one of them gets a good estimate.

- **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 $\text{increment}(x)$ in the overall structure, we call $\text{increment}(x)$ 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 $\hat{a}_i$ is the sum of two independent terms:

- The actual value $a_i$.
- Some “noise” terms from other elements colliding with $x_i$.

Since the noise terms are always nonnegative, larger values of $\hat{a}_i$ are less accurate than smaller values of $\hat{a}_i$.

**Idea:** Take, as our estimate, the minimum value of $\hat{a}_i$ from all of the data structures.
The Final Analysis

• For each independent copy of this data structure, the probability that our estimate is within $\varepsilon ||a||_1$ of the true value is at least $1 - \frac{1}{e}$.

• Let $\mathcal{E}_i$ be the event that the $i$th copy of the data structure provides an estimate within $\varepsilon ||a||_1$ of the true answer.

• Let $\mathcal{E}$ be the event that the aggregate data structure provides an estimate within $\varepsilon ||a||_1$.

• **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,
  \[ \Pr[\varepsilon] = \Pr[\exists i. \varepsilon_i] \]
• Equivalently:
  \[ \Pr[\varepsilon] = 1 - \Pr[\forall i. \overline{\varepsilon}_i] \]
• Since all the estimates are independent:
  \[ \Pr[\varepsilon] = 1 - \Pr[\forall i. \overline{\varepsilon}_i] \geq 1 - \frac{1}{e^d}. \]
The Final Analysis

- We now have that
  \[ \Pr[\mathcal{E}] \geq 1 - \frac{1}{e^d}. \]
- If we want the confidence to be \( 1 - \delta \), we can choose \( \delta \) such that
  \[ 1 - \delta = 1 - \frac{1}{e^d} \]
- 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 \( \varepsilon \|a\|_1 \) is at least \( 1 - \delta \).
The Count-Min Sketch

- This data structure is called the \textit{count-min sketch}.
- Given parameters $\varepsilon$ and $\delta$, choose
  $$w = \lceil e / \varepsilon \rceil \quad d = \lceil \ln \delta^{-1} \rceil$$
- Create an array \texttt{count} of size $w \times d$ and for each row $i$, choose a hash function $h_i : \mathcal{U} \rightarrow [w]$ uniformly and independently from a 2-independent family of hash functions $\mathcal{H}$.
- To \texttt{increment}(x), increment \texttt{count}[i][h_i(x)] for each row $i$.
- To \texttt{estimate}(x), return the minimum value of \texttt{count}[i][h_i(x)] across all rows $i$. 
The Count-Min Sketch

- Update and query times are $\Theta(d)$, which is $\Theta(\log \delta^{-1})$.

- Space usage: $\Theta(\varepsilon^{-1} \cdot \log \delta^{-1})$ counters.
  - This can be *significantly* better than just storing a raw frequency count!

- Provides an estimate to within $\varepsilon \| \mathbf{a} \|_1$ with probability at least $1 - \delta$. 

Some Generalizable Ideas

- Many of the techniques and ideas from this analysis will show up in other places.
- First, the idea of using indicator variables and linearity of expectation to simplify expected value calculations.
- Second, relying on the independence guarantees of our hash function to simplify some of the intermediate steps.
- Third, the fact that being good on expectation isn’t the same as being good with high probability and using concentration inequalities to quantify spread.
- Finally, the fact that confidence and accuracy aren’t the same, and running multiple parallel copies of a data structure to boost confidence.
Time-Out for Announcements!
Final Project Proposal

- Final project proposals were due today at 2:30PM.
- We’re going to run a matchmaking algorithm soon and get back to everyone with their assigned topics.
- We’re looking forward to seeing what everyone has come up with!
Problem Sets

- Problem Set Four is due next Thursday at 2:30PM.
- Have questions? As always, you can
  - stop by office hours, or
  - ask on Piazza!
- We hope you have fun with this one!
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 \( \text{count} \) of length \( w \).
- As before, choose a hash function \( h : \mathcal{U} \rightarrow [w] \) from a family \( \mathcal{H} \).
- For each \( x_i \in \mathcal{U} \), assign \( x_i \) either +1 or -1.
- To \textit{increment}(x), go to \( \text{count}[h(x)] \) and add ±1 as appropriate.
- To \textit{estimate}(x), return \( \text{count}[h(x)] \), 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 have $h \in \mathcal{H}$ chosen uniformly at random from a 3-independent family of hash functions from $\mathcal{U}$ to $\mathcal{W}$.
- Choose $s \in \mathcal{U}$ uniformly randomly and independently of $h$ from a 3-independent family from $\mathcal{U}$ to $\{-1, +1\}$.
  - (Note: The more traditional analysis uses 2-independence rather than 3-independence. I’m showing you a slightly simplified version.)
- 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 $\hat{a}_i$ to be our estimate of $a_i$.
• As before, $\hat{a}_i$ 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

$$s(x_i) \cdot s(x_j) \cdot a_j$$

• Why?
  • The counter for $x_i$ will have $s(x_j) \cdot a_j$ added in.
  • We multiply the counter by $s(x_i)$ before returning it.
Formalizing the Intuition

● As before, define $\hat{a}_i$ to be our estimate of $a_i$.

● As before, $\hat{a}_i$ 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

$$s(x_i) \cdot s(x_j) \cdot a_j$$

● Or:

  ● If $s(x_i)$ and $s(x_j)$ point in the same direction, the terms add to the total.
  
  ● If $s(x_i)$ and $s(x_j)$ point in different directions, the terms subtract from the total.
Formalizing the Intuition

- In our quest to learn more about $\hat{a}_i$, let's have $X_j$ be a random variable indicating whether $x_i$ and $x_j$ collided with one another:

$$X_j = \begin{cases} 
1 & \text{if } h(x_i) = h(x_j) \\
0 & \text{if } h(x_i) \neq h(x_j)
\end{cases}$$

- We can then express $\hat{a}_i$ in terms of the signed contributions from the items it collides with:

$$\hat{a}_i = \sum_j a_j s(x_i) s(x_j) X_j = a_i + \sum_{j \neq i} a_j s(x_i) s(x_j) X_j$$

This is how much the collision impacts our estimate.

We only care about items we collided with.
\[ E[\hat{a}_i] = E[a_i + \sum_{j \neq i} a_j s(x_i) s(x_j) X_j] \]

\[ = E[a_i] + E[\sum_{j \neq i} a_j s(x_i) s(x_j) X_j] \]

Hey, it’s linearity of expectation!
\[
\mathbb{E}[\hat{a}_i] = \mathbb{E}[a_i + \sum_{j \neq i} a_j s(x_i) s(x_j) X_j] \\
= \mathbb{E}[a_i] + \mathbb{E}[\sum_{j \neq i} a_j s(x_i) s(x_j) X_j] \\
= a_i + \sum_{j \neq i} \mathbb{E}[a_j s(x_i) s(x_j) X_j]
\]

Remember that \(a_i\) and the like aren’t random variables.
\[ E[\hat{a}_i] = E[a_i + \sum_{j \neq i} a_j s(x_i)s(x_j)X_j] \]
\[ = E[a_i] + E[\sum_{j \neq i} a_j s(x_i)s(x_j)X_j] \]
\[ = a_i + \sum_{j \neq i} E[a_j s(x_i)s(x_j)X_j] \]
\[ = a_i + \sum_{j \neq i} E[s(x_i)s(x_j)]E[a_j X_j] \]

We chose the hash functions \( h \) and \( s \) independently of one another.

\[ X_j = \begin{cases} 
1 & \text{if } h(x_i) = h(x_j) \\
0 & \text{if } h(x_i) \neq h(x_j) 
\end{cases} \]
\[ \mathbb{E}[\hat{a}_i] = \mathbb{E}[a_i + \sum_{j \neq i} a_j s(x_i) s(x_j) X_j] \]

\[ = \mathbb{E}[a_i] + \mathbb{E}[\sum_{j \neq i} a_j s(x_i) s(x_j) X_j] \]

\[ = a_i + \sum_{j \neq i} \mathbb{E}[a_j s(x_i) s(x_j) X_j] \]

\[ = a_i + \sum_{j \neq i} \mathbb{E}[s(x_i) s(x_j)] \mathbb{E}[a_j X_j] \]

\[ = a_i + \sum_{j \neq i} \mathbb{E}[s(x_i)] \mathbb{E}[s(x_j)] \mathbb{E}[a_j X_j] \]

Remember that \( s \) is drawn from a 3-independent family of hash functions, so \( s(x_i) \) and \( s(x_j) \) are independent random variables.
\[
\begin{align*}
\mathbb{E}[\hat{a}_i] &= \mathbb{E}[a_i + \sum_{j \neq i} a_j s(x_i) s(x_j) X_j] \\
&= \mathbb{E}[a_i] + \mathbb{E}\left[\sum_{j \neq i} a_j s(x_i) s(x_j) X_j\right] \\
&= a_i + \sum_{j \neq i} \mathbb{E}[a_j s(x_i) s(x_j) X_j] \\
&= a_i + \sum_{j \neq i} \mathbb{E}[s(x_i) s(x_j)] \mathbb{E}[a_j X_j] \\
&= a_i + \sum_{j \neq i} \mathbb{E}[s(x_i)] \mathbb{E}[s(x_j)] \mathbb{E}[a_j X_j] \\
&= a_i + \sum_{j \neq i} 0 \\
&= a_i
\end{align*}
\]

\[
\mathbb{E}[s(x_i)] = \frac{1}{2} \cdot (-1) + \frac{1}{2} \cdot (+1) = 0
\]

s is drawn from a 3-independent family of hash functions.

s(x_i) is uniform over \{-1, +1\}

Pr[s(x_i) = -1] = \frac{1}{2} \quad \text{Pr}[s(x_i) = +1] = \frac{1}{2}
Expecting the Unexpected

• We’ve just seen that $E[\hat{a}_i] = a_i$, so on expectation our estimate is perfectly correct!

• However, we have no idea how likely it is that we’re going to get an estimate like this.

• Let’s see if we can bound the likelihood that we stray far from $a_i$. 
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 we had a one-sided error: the distance $\hat{a}_i - a_i$ from the true answer was nonnegative.

• However, with the count sketch, we have a two-sided error: $\hat{a}_i - a_i$ can be negative in the count sketch because collisions can decrease the estimate $\hat{a}_i$ below the true value $a_i$.

• 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:

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

- Equivalently:

$$
\Pr\left[ |X - \mathbb{E}[X]| \geq c \right] \leq \frac{\text{Var}[X]}{c^2}
$$

- If we can get the variance of $\hat{a}_i$, we can bound the probability that we get a bad estimate with our data structure.
Let’s try computing the variance of our estimate $\hat{a}_i$:

$$\text{Var}[\hat{a}_i] = \text{Var}[a_i + \sum_{j \neq i} a_j s(x_i) s(x_j) X_j]$$

$$= \text{Var} \left[ \sum_{j \neq i} a_j s(x_i) s(x_j) X_j \right]$$

$$\text{Var}[a + X] = \text{Var}[X]$$
Computing the Variance

- Let’s try computing the variance of our estimate \( \hat{a}_i \):

\[
\text{Var}[\hat{a}_i] = \text{Var}[a_i + \sum_{j \neq i} a_j s(x_i)s(x_j)X_j]
\]

\[
= \text{Var}[\sum_{j \neq i} a_j s(x_i)s(x_j)X_j]
\]

- Variance is not a linear operator, but it is linear if the underlying random variables are independent of one another.

- **Claim**: Each term of the sum is independent of the others.
We want to show that these two terms are independent:

\[ a_j s(x_i) s(x_j) X_j \quad a_k s(x_i) s(x_k) X_k \]

Imagine we know \( a_j s(x_i) s(x_j) X_j \).

Whether \( a_k s(x_i) s(x_k) X_k = 0 \) depends on whether \( h(x_i) = h(x_k) \).

- The values \( h(x_i), h(x_j), \) and \( h(x_k) \) are uniformly-random and independent because \( h \) is 3-independent.
- Knowing whether \( h(x_i) = h(x_j) \) doesn’t impact the probability that \( h(x_i) = h(x_k) \), since all three values are uniform and independent.

The sign of \( a_k s(x_i) s(x_k) X_k \) depends on \( s(x_i) \cdot s(x_k) \).

- \( s(x_i), s(x_j), \) and \( s(x_k) \) are uniformly-random and independent because \( s \) is 3-independent.
- There’s an equal chance that \( s(x_i) \cdot s(x_k) = 1 \) and \( s(x_i) \cdot s(x_k) = -1 \), since even with \( s(x_i) \cdot s(x_j) \) fixed, \( s(x_k) \) is independently and uniformly distributed over \( \{+1, -1\} \).
\[\text{Var}[\hat{a}_i] = \text{Var}[a_i + \sum_{j \neq i} a_j s(x_i) s(x_j) X_j]\]
\[= \text{Var}\left[\sum_{j \neq i} a_j s(x_i) s(x_j) X_j\right]\]
\[= \sum_{j \neq i} \text{Var}[a_j s(x_i) s(x_j) X_j]\]
\[\leq \sum_{j \neq i} E[(a_j s(x_i) s(x_j) X_j)^2]\]

\[\text{Var}[Z] = E[Z^2] - E[Z]^2 \leq E[Z^2]\]
\[
\text{Var}[\hat{a}_i] = \text{Var}[a_i + \sum_{j \neq i} a_j s(x_i) s(x_j) X_j]
\]
\[
= \text{Var}[\sum_{j \neq i} a_j s(x_i) s(x_j) X_j]
\]
\[
= \sum_{j \neq i} \text{Var}[a_j s(x_i) s(x_j) X_j]
\]
\[
\leq \sum_{j \neq i} E[(a_j s(x_i) s(x_j) X_j)^2]
\]
\[
= \sum_{j \neq i} E[a_j^2 s(x_i)^2 s(x_j)^2 X_j^2]
\]
\[
= \sum_{j \neq i} a_j^2 E[X_j^2]
\]

\[
\begin{align*}
\text{s(x)} &= \pm 1, \\
\text{so} \\
\text{s(x)}^2 &= 1
\end{align*}
\]
Var[\hat{a}_i] = Var[a_i + \sum_{j \neq i} a_j s(x_i) s(x_j) X_j] \\
= Var[\sum_{j \neq i} a_j s(x_i) s(x_j) X_j] \\
= \sum_{j \neq i} Var[a_j s(x_i) s(x_j) X_j] \\
\leq \sum_{j \neq i} E[(a_j s(x_i) s(x_j) X_j)^2] \\
= \sum_{j \neq i} E[a_j^2 s(x_i)^2 s(x_j)^2 X_j^2] \\
= \sum_{j \neq i} a_j^2 E[X_j^2] \\
= \sum_{j \neq i} a_j^2 E[X_j] \\

**Useful Fact:**
If X is an indicator variable, then $X^2 = X$. 
\[
\text{Var}[\hat{a}_i] = \text{Var}[a_i + \sum_{j \neq i} a_j s(x_i) s(x_j) X_j]
\]

\[
= \text{Var}\left[\sum_{j \neq i} a_j s(x_i) s(x_j) X_j\right]
\]

\[
= \sum_{j \neq i} \text{Var}[a_j s(x_i) s(x_j) X_j]
\]

\[
\leq \sum_{j \neq i} \mathbb{E}\left[(a_j s(x_i) s(x_j) X_j)^2\right]
\]

\[
= \sum_{j \neq i} \mathbb{E}\left[a_j^2 s(x_i)^2 s(x_j)^2 X_j^2\right]
\]

\[
= \sum_{j \neq i} a_j^2 \mathbb{E}[X_j^2]
\]

\[
= \sum_{j \neq i} a_j^2 \mathbb{E}[X_j]
\]

\[
= \sum_{j \neq i} a_j^2 / w
\]

\[
X_j = \begin{cases} 
1 & \text{if } h(x_i) = h(x_j) \\
0 & \text{if } h(x_i) \neq h(x_j)
\end{cases}
\]
\[
\text{Var}[\hat{a}_i] = \text{Var}[a_i + \sum_{j \neq i} a_j s(x_i) s(x_j) X_j] \\
= \text{Var}\left[\sum_{j \neq i} a_j s(x_i) s(x_j) X_j\right] \\
= \sum_{j \neq i} \text{Var}[a_j s(x_i) s(x_j) X_j] \\
\leq \sum_{j \neq i} \mathbb{E}\left[(a_j s(x_i) s(x_j) X_j)^2\right] \\
= \sum_{j \neq i} \mathbb{E}[a_j^2 s(x_i)^2 s(x_j)^2 X_j^2] \\
= \sum_{j \neq i} a_j^2 \mathbb{E}[X_j^2] \\
= \sum_{j \neq i} a_j^2 \mathbb{E}[X_j] \\
= \sum_{j \neq i} a_j^2 / w \\
\leq \left\|a\right\|_2^2 / w
\]

\[\sqrt{\sum_j a_j^2} = \|a\|_2\]
\[
\text{Var}[\hat{a}_i] = \text{Var}[a_i + \sum_{j \neq i} a_j s(x_i) s(x_j) X_j]
\]
\[
= \text{Var}[\sum_{j \neq i} a_j s(x_i) s(x_j) X_j]
\]
\[
= \sum_{j \neq i} \text{Var}[a_j s(x_i) s(x_j) X_j]
\]
\[
\leq \sum_{j \neq i} \mathbb{E}[(a_j s(x_i) s(x_j) X_j)^2]
\]
\[
= \sum_{j \neq i} \mathbb{E}[a_j^2 s(x_i)^2 s(x_j)^2 X_j^2]
\]
\[
= \sum_{j \neq i} a_j^2 \mathbb{E}[X_j^2]
\]
\[
= \sum_{j \neq i} a_j^2 \mathbb{E}[X_j]
\]
\[
= \sum_{j \neq i} a_j^2 / w
\]
\[
\leq \|a\|_2^2 / w
\]

I know this might look really dense, but many of these substeps end up being really useful techniques. These ideas generalize, I promise.
Harnessing Chebyshev

- Chebyshev's Inequality says
  \[ \Pr\left[ |X - E[X]| \geq c \sqrt{\text{Var}[X]} \right] \leq \frac{1}{c^2} \]

- Applying this to \( \hat{a}_i \) yields
  \[ \Pr\left[ |\hat{a}_i - a_i| \geq \frac{c \|a\|_2}{\sqrt{w}} \right] \leq \frac{1}{c^2} \]

- Given error parameter \( \varepsilon \), pick \( w = \lceil e / \varepsilon^2 \rceil \), so
  \[ \Pr\left[ |\hat{a}_i - a_i| \geq \frac{c \varepsilon \|a\|_2}{\sqrt{e}} \right] \leq \frac{1}{c^2} \]

- Therefore, choosing \( c = e^{1/2} \) gives
  \[ \Pr\left[ |\hat{a}_i - a_i| \geq \varepsilon \|a\|_2 \right] \leq \frac{1}{e} \]
The Story So Far

- We now know that, by setting $\varepsilon = (e / w)^{1/2}$, the estimate is within $\varepsilon \| a \|_2$ 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 $\varepsilon \| a \|_2$ rather than $\varepsilon \| a \|_1$.

- 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 $\varepsilon \|a\|_2$ is if at least half of the data structures output an answer that is more than $\varepsilon \|a\|_2$ 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 $\varepsilon \|a\|_2$ 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

\[
\Pr[ X > \frac{d}{2} ] \leq e^{-\frac{-d(1/2-1/e)^2}{2(1/e)}} = e^{-k\cdot d} \quad \text{(for some constant } k)\]

• Therefore, choosing \( d = k^{-1} \cdot \log \delta^{-1} \) 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

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

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

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

The specific constant factor here matters, since it’s an exponent! To implement this data structure, you’ll need to work out the exact value.

Therefore, choosing $d = k^{-1} \cdot \log \delta^{-1}$ 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 $\varepsilon$ and $\delta$, choose
  \[ w = \lceil e / \varepsilon^2 \rceil \quad d = \Theta(\log \delta^{-1}) \]
- 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 $s_i(x) \cdot \text{count}[i][h_i(x)]$ for each row $i$. 
The Final Analysis

- With probability at least $1 - \delta$, all estimates are accurate to within a factor of $\varepsilon \|a\|_2$.
- Space usage is $\Theta(w \times d)$, which we've seen to be $\Theta(\varepsilon^{-2} \cdot \log \delta^{-1})$.
- Updates and queries run in time $\Theta(\delta^{-1})$.
- Trades factor of $\varepsilon^{-1}$ space for an accuracy guarantee relative to $\|a\|_2$ versus $\|a\|_1$. 
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 $a_i$ terms individually we want to estimate $\|a\|_1$ or $\|a_2\|$, we can use a **frequency moment estimator**.

- You’ll get to play around with at least one of these on Problem Set Five.
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!