Frequency Estimators
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.
- These data structures are used *extensively* in practice. Each of the next four lectures is on something you’re likely to encounter IRL.
- Each of the next four lectures explores powerful techniques that are useful in navigating the rivers of Theoryland.
Where We’re Going

- **Frequency Estimation (Today)**
  - Can we count quantities without actually counting them?

- **Hash Tables (Tuesday / Thursday)**
  - Everyone agrees these are good ideas. How do you design fast hash tables, and why are they fast?

- **Approximate Membership (Next Tuesday)**
  - Squeezing as much value from our bits as possible.
Outline for Today

- **Hash Functions**
  - Understanding our basic building blocks.

- **Count-Min Sketches**
  - Estimating how many times we’ve seen something.

- **Concentration Inequalities**
  - “Correct on expectation” versus “correct with high probability.”

- **Probability Amplification**
  - Increasing our confidence in our answers.

- **Count Sketches**
  - These ideas transfer well. Here’s another example.
Preliminaries: 2-Independent Hashing
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 $[m] = \{0, 1, 2, 3, ..., m-1\}$

$$h : \mathcal{U} \rightarrow [m]$$
Families of Hash Functions

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

- We’ll usually sample hash functions uniformly and independently from a family as needed.

- **Key point:** The randomness in our data structures almost always derives from the random choice of hash functions, not from the data.

- **Question:** What makes a family of hash functions $\mathcal{H}$ a “good” family of hash functions?
Goal: If we pick \( h \in \mathcal{H} \) uniformly at random, then \( h \) should distribute elements uniformly randomly.

Problem: Representing a hash function for a sample of \( n \) elements from \( \mathcal{U} \) requires \( \Omega(n \log m) \) bits.

Question: Do we actually need true randomness? Or can we get away with something weaker?
**Distribution Property:**
Each element should have an equal probability of being placed in each slot.

**Problem:** This rule doesn’t guarantee that elements are spread out.

For any $x \in \mathcal{U}$ and random $h \in \mathcal{H}$, the value of $h(x)$ is uniform over $[m]$. 
**Distribution Property:**
Each element should have an equal probability of being placed in each slot.

**Independence Property:**
Where one element is placed shouldn’t impact where a second goes.

For any \( x \in \mathcal{U} \) and random \( h \in \mathcal{H} \), the value of \( h(x) \) is uniform over \([m]\).

For any distinct \( x, y \in \mathcal{U} \) and random \( h \in \mathcal{H} \), \( h(x) \) and \( h(y) \) are independent random variables.
**Distribution Property:**
Each element should have an equal probability of being placed in each slot.

**Independence Property:**
Where one element is placed shouldn’t impact where a second goes.

For any $x \in \mathcal{U}$ and random $h \in \mathcal{H}$, the value of $h(x)$ is uniform over $[m]$.

For any distinct $x, y \in \mathcal{U}$ and random $h \in \mathcal{H}$, $h(x)$ and $h(y)$ are independent random variables.

A family of hash functions $\mathcal{H}$ is called **2-independent** (or **pairwise independent**) if it satisfies the distribution and independence properties.
For any $x \in \mathcal{U}$ and random $h \in \mathcal{H}$, the value of $h(x)$ is uniform over $[m]$.

**Intuition:**
2-independence means any pair of elements is unlikely to collide.

For any distinct $x, y \in \mathcal{U}$ and random $h \in \mathcal{H}$, $h(x)$ and $h(y)$ are independent random variables.

\[
\Pr[h(x) = h(y)] = \sum_{i=0}^{m-1} \Pr[h(x) = i \land h(y) = i] = \sum_{i=0}^{m-1} \Pr[h(x) = i] \cdot \Pr[h(y) = i]
\]

Graph:
```
\[\text{y
\[\text{x
\]
0 1 2 ... m-1
```
For any $x \in \mathcal{U}$ and random $h \in \mathcal{H}$, the value of $h(x)$ is uniform over $[m]$.

**Intuition:**
2-independence means any pair of elements is unlikely to collide.

$$
\Pr[h(x) = h(y)] = \sum_{i=0}^{m-1} \Pr[h(x) = i \land h(y) = i]
$$

$$
= \sum_{i=0}^{m-1} \Pr[h(x) = i] \cdot \Pr[h(y) = i]
$$

$$
= \sum_{i=0}^{m-1} \frac{1}{m^2}
$$
For any \( x \in \mathcal{U} \) and random \( h \in \mathcal{H} \), the value of \( h(x) \) is uniform over \([m]\).

*Intuition:* 2-independence means any pair of elements is unlikely to collide.

\[
\Pr[h(x) = h(y)] = \sum_{i=0}^{m-1} \Pr[h(x) = i] \land h(y) = i
\]

\[
= \sum_{i=0}^{m-1} \Pr[h(x) = i] \cdot \Pr[h(y) = i]
\]

\[
= \sum_{i=0}^{m-1} \frac{1}{m^2}
\]

\[
= \frac{1}{m}
\]

This is the same as if \( h \) were a truly random function.
For more on hashing outside of Theoryland, check out this Stack Exchange post.
Approximating Quantities
What makes for a good “approximate” solution?
What does it mean for an approximation to be “good”?

Let $A$ be the true answer. Let $\hat{A}$ be a random variable denoting our estimate.

This would not make for a good estimate. However, we have $E[\hat{A}] = A$.

**Observation 1:** Being correct in expectation isn’t sufficient.
Let $A$ be the true answer. Let $\hat{A}$ be a random variable denoting our estimate.

It’s unlikely that we’ll get the right answer, but we’re probably going to be close.

**Observation 2:** The difference $|\hat{A} - A|$ between our estimate and the truth should ideally be small.

What does it mean for an approximation to be “good”? 
What does it mean for an approximation to be “good”?

Let $A$ be the true answer. Let $\hat{A}$ be a random variable denoting our estimate.

This estimate skews low, but it’s very close to the true value.

**Observation 3:** An estimate doesn’t have to be unbiased to be useful.
Let $A$ be the true answer. Let $\hat{A}$ be a random variable denoting our estimate. The more resources we allocate, the better our estimate should be. 

**Observation 4:** A good approximation should be tunable.

What does it mean for an approximation to be “good”?
Suppose there are two tunable values

\[ \varepsilon \in (0, 1] \]
\[ \delta \in (0, 1] \]

where \( \varepsilon \) represents \textit{accuracy} and \( \delta \) represents \textit{confidence}.

**Goal:** Make an estimator \( \hat{A} \) for some quantity \( A \) where

\[
\text{With probability at least } 1 - \delta, \quad |\hat{A} - A| \leq \varepsilon \cdot \text{size(input)}
\]

for some measure of the size of the input.

What does it mean for an approximation to be “good”? 
**Goal:** Make an estimator $\hat{A}$ for some quantity $A$ where

With probability at least $1 - \delta$,

$$|A - \hat{A}| \leq \varepsilon \cdot \text{size(input)}$$

for some measure of the size of the input.

$\delta = \frac{1}{2}$

$\varepsilon$ small

---

What does it mean for an approximation to be “good”?
What does it mean for an approximation to be “good”?

**Goal:** Make an estimator \( \hat{A} \) for some quantity \( A \) where

\[
\text{With probability at least } 1 - \delta, \quad |A - \hat{A}| \leq \varepsilon \cdot \text{size}(\text{input})
\]

for some measure of the size of the input.

\( \delta = \frac{1}{2} \)

\( \varepsilon \) is medium
**Goal:** Make an estimator $\hat{A}$ for some quantity $A$ where

With probability at least $1 - \delta$,

$$|A - \hat{A}| \leq \varepsilon \cdot \text{size(input)}$$

for some measure of the size of the input.

$\delta = \frac{1}{2}$
$\varepsilon$ large

What does it mean for an approximation to be “good”?
What does it mean for an approximation to be “good”?

**Goal:** Make an estimator \( \hat{A} \) for some quantity \( A \) where

\[
\text{With probability at least } 1 - \delta, \quad |A - \hat{A}| \leq \varepsilon \cdot \text{size}(\text{input})
\]

for some measure of the size of the input.

\( \delta = \frac{1}{2} \)
\( \varepsilon \) small
What does it mean for an approximation to be “good”?

Goal: Make an estimator $\hat{A}$ for some quantity $A$ where

With probability at least $1 - \delta$,

$|A - \hat{A}| \leq \varepsilon \cdot \text{size(input)}$

for some measure of the size of the input.

$\delta = \frac{1}{4}$

$\varepsilon$ small

Correct

Approximately

Probably

True answer
What does it mean for an approximation to be “good”?

Goal: Make an estimator \( \hat{A} \) for some quantity \( A \) where

\[
\begin{align*}
\text{With probability at least } 1 - \delta, \quad |A - \hat{A}| &\leq \varepsilon \cdot \text{size(input)} \quad \text{(Approximately Correct)}
\end{align*}
\]

for some measure of the size of the input.

\( \delta = \frac{1}{16} \)
\( \varepsilon \) small

True answer
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.
The Count-Min Sketch
How to Build an Estimator

1. Design a simple data structure that, intuitively, gives you a good estimate.

2. Use a *sum of indicator variables* and *linearity of expectation* to prove that, on expectation, the data structure is pretty close to correct.

3. Use a *concentration inequality* to show that the data structure’s output is close to its expectation.

4. Run multiple copies of the data structure in parallel to amplify the success probability.
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 $x$.

![Diagram showing counters and values]
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 $\text{count}[h(x)]$. 

```
+-----+   +-----+   +-----+   +-----+   +-----+   +-----+
| 137 |   |  42 |   | 166 |   |     |   | 161 |
```

$\chi \xrightarrow{h} \text{count}$
Analyzing our Structure
For each $x_i \in \mathcal{U}$, let $a_i$ denote the number of times we’ve seen $x_i$.

Similarly, let $\hat{a}_i$ denote our estimated value of the frequency of $x_i$.

**Goal:** Show that the error in our estimate $(\hat{a}_i - a_i)$ is probably close to zero.
**Idea:** Think of our element frequencies \( a_1, a_2, a_3, \ldots \) as a vector \( a = [a_1, a_2, a_3, \ldots] \).

The total number of objects is the sum of the vector entries.

This is called the \( L_1 \) norm of \( a \), and is denoted \( \| a \|_1 \):

\[
\| a \|_1 = \sum_i |a_i|
\]

There are \( \| a \|_1 \) total elements distributed across \( w \) buckets. We’re using a 2-independent hash family.

**Reasonable guess:** each bin has \( \| a \|_1 / w \) elements in it, so \( \hat{a}_i - a_i \leq \| a \|_1 / w \)

**Question:** Intuitively, what should we expect our approximation error to be?
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 $\hat{a}_i - a_i$ is then given by

$$\hat{a}_i - a_i = \sum_{j \neq i} a_j X_j$$
\[ \mathbb{E}[\hat{a}_i - a_i] = \mathbb{E}\left[ \sum_{j \neq i} a_j X_j \right] \]

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

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

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

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

The values of \(a_j\) are not random. The randomness comes from our choice of hash function.
\[ E[\hat{a}_i - a_i] = E \left[ \sum_{j \neq i} a_j X_j \right] \]
\[ = \sum_{j \neq i} E[a_j X_j] \]
\[ = \sum_{j \neq i} a_j E[X_j] \]

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

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

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

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!
\[ E[\hat{a}_i - a_i] = E\left[ \sum_{j \neq i} a_j X_j \right] \]
\[ = \sum_{j \neq i} E[a_j X_j] \]
\[ = \sum_{j \neq i} a_j E[X_j] \]
\[ = \sum_{j \neq i} \frac{a_j}{W} \]

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

Hey, we saw this earlier!
\[
E[\hat{a}_i - a_i] = E[\sum_{j \neq i} a_j X_j] = \sum_{j \neq i} E[a_j X_j] = \sum_{j \neq i} a_j E[X_j] = \sum_{j \neq i} \frac{a_j}{w} \leq \frac{||a||_1}{w}
\]

\[
E[X_j] = 1 \cdot \Pr[h(x_i) = h(x_j)] + 0 \cdot \Pr[h(x_i) \neq h(x_j)] = 1 \cdot \Pr[h(x_i) = h(x_j)] = \frac{1}{w}
\]
**Goal:** Make an estimator \( \hat{a} \) for some quantity \( a \) where

\[
\text{With probability at least } 1 - \delta, \quad |\hat{a} - a| \leq \varepsilon \cdot \text{size(input)}
\]

for some measure of the size of the input.

How do we tune \( w \) so we’re likely to fall in this range?

\[
E[\hat{a}_i - a_i] \leq \frac{\|a\|_1}{w}
\]
$$\Pr \left[ \hat{a}_i - a_i > \varepsilon \|a\|_1 \right]$$

$$< \frac{\mathbb{E} \left[ \hat{a}_i - a_i \right]}{\varepsilon \|a\|_1}$$

We don’t know the exact distribution of this random variable.

However, we have a one-sided error: our estimate can never be lower than the true value. This means that $\hat{a}_i - a_i \geq 0$.

Markov’s inequality says that if $X$ is a nonnegative random variable, then

$$\Pr [ X > c ] < \frac{\mathbb{E} [ X ]}{c}.$$
Pr \[\hat{a}_i - a_i > \varepsilon \|a\|_1\]

\[
< \frac{E[\hat{a}_i - a_i]}{\varepsilon \|a\|_1}
\]

\[
\leq \frac{\|a\|_1 \cdot 1}{w \varepsilon \|a\|_1}
\]

\[
E[\hat{a}_i - a_i] \leq \frac{\|a\|_1}{w}
\]
\[
\Pr[\hat{a}_i - a_i > \varepsilon \|a\|_1] \\
\leq \frac{\mathbb{E}[\hat{a}_i - a_i]}{\varepsilon \|a\|_1} \\
\leq \frac{\|a\|_1}{w} \cdot \frac{1}{\varepsilon \|a\|_1} \\
= \frac{1}{\varepsilon w}
\]
**Goal:** Make an estimator $\hat{a}$ for some quantity $a$ where

With probability at least $1 - \delta$,

$$|\hat{a} - a| \leq \varepsilon \cdot \text{size}(\text{input})$$

for some measure of input size.

Pr[$\hat{a}_i - a_i > \varepsilon \|a\|_1$] \leq \frac{1}{\varepsilon w}$

**Initial Idea:**
Pick $w = \varepsilon^{-1} \cdot \delta^{-1}$. Then

$\Pr[\hat{a}_i - a_i > \varepsilon \|a\|_1] < \delta$

**Correct**
Probably
Approximately

Suppose we’re counting 1,000 distinct items.

If we want our estimate to be within $\varepsilon \|a\|_1$ of the true value with 99.9% probability, how much memory do we need?

**Answer:** $1,000 \cdot \varepsilon^{-1}$.

Can we do better?
**Goal:** Make an estimator \( \hat{a} \) for some quantity \( a \) where

With probability at least \( 1 - \delta \),

\[
|\hat{a} - a| \leq \varepsilon \cdot \text{size}(\text{input})
\]

for some measure of input size.

\[
\Pr[\hat{a}_i - a_i > \varepsilon \|a\|_1] \leq \frac{1}{\varepsilon w}
\]

**Revised Idea:** Pick \( w = e \cdot \varepsilon^{-1} \). Then

\[
\Pr[\hat{a}_i - a_i > \varepsilon \|a\|_1] < e^{-1}
\]

This simple data structure, by itself, is likely to be wrong.

What happens if we run a bunch of copies of this approach in parallel?
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?

<table>
<thead>
<tr>
<th>Estimator 1:</th>
<th>Estimator 2:</th>
<th>Estimator 3:</th>
<th>Estimator 4:</th>
<th>Estimator 5:</th>
</tr>
</thead>
<tbody>
<tr>
<td>137</td>
<td>271</td>
<td>166</td>
<td>103</td>
<td>261</td>
</tr>
</tbody>
</table>
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.
Recognizing the Answer

• Suppose we have \( d \) independent copies of our estimator.

• Let \( \hat{a}_{ij} \) be the estimate returned by the \( j \)th copy of the estimator.

• Our overall estimate is therefore

\[
\min \{ \hat{a}_{ij} \}
\]

• **Question:** How likely is this to be within our magic window around the true value?
\[ \Pr \left[ \min \{ \hat{a}_{ij} \} - a_i > \varepsilon \|a\|_1 \right] \]

\[ = \Pr \left[ \bigwedge_j \left( \hat{a}_{ij} - a_i > \varepsilon \|a\|_1 \right) \right] \]

The only way the minimum estimate is inaccurate is if every estimate is inaccurate.

Let \( \hat{a}_{ij} \) be the estimate from the \( j \)th copy of the data structure.

Our final estimate is \( \min \{ \hat{a}_{ij} \} \).
\[
\Pr \left[ \min \{ \hat{a}_{ij} \} - a_i > \varepsilon \|a\|_1 \right]
\]

\[
= \Pr \left[ \bigwedge_j \left( \hat{a}_{ij} - a_i > \varepsilon \|a\|_1 \right) \right]
\]

\[
= \prod_j \Pr \left[ \hat{a}_{ij} - a_i > \varepsilon \|a\|_1 \right]
\]

Each copy of the data structure is independent of the others.

Let \( \hat{a}_{ij} \) be the estimate from the \( j \)th copy of the data structure.

Our final estimate is \( \min \{ \hat{a}_{ij} \} \).
\[
\Pr \left[ \min \{ \hat{a}_{ij} \} - a_i > \varepsilon \|a\|_1 \right] \\
= \Pr \left[ \bigwedge_{j} (\hat{a}_{ij} - a_i > \varepsilon \|a\|_1) \right] \\
= \prod_{j} \Pr [\hat{a}_{ij} - a_i > \varepsilon \|a\|_1] \\
< \prod_{j} e^{-1}
\]

Let \( \hat{a}_{ij} \) be the estimate from the \( j \)th copy of the data structure.

Our final estimate is \( \min \{ \hat{a}_{ij} \} \)
Let $\hat{a}_{ij}$ be the estimate from the $j$th copy of the data structure.

Our final estimate is $\min \{ \hat{a}_{ij} \}$.
**Goal:** Make an estimator $\hat{a}$ for some quantity $a$ where

With probability at least $1 - \delta$,

$|\hat{a} - a| \leq \varepsilon \cdot \text{size}(\text{input})$

for some measure of input size.

**Idea:** Choose $d = -\ln \delta$. (Equivalently: $d = \ln \delta^{-1}$.) Then

$\Pr \left[ \min \{ \hat{a}_{ij} - a_i \} > \varepsilon \|a\|_1 \right] < e^{-d}$

$\Pr \left[ \min \{ \hat{a}_{ij} - a_i \} > \varepsilon \|a\|_1 \right] < \delta$
The Count-Min Sketch

- This data structure is called the *count-min sketch*.
- Given parameters $\varepsilon$ and $\delta$, choose
  \[ w = \lceil e / \varepsilon \rceil \quad d = \lceil \ln \delta^{-1} \rceil \]
- Create an array `count` of size $w \times d$ and for each row $i$, choose a hash function $h_i : \mathcal{U} \rightarrow [w]$ uniformly and independently from a 2-independent family of hash functions $\mathcal{H}$.
- 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 $\Theta(d)$, which is $\Theta(\log \delta^{-1})$.
- Space usage: $\Theta(\varepsilon^{-1} \cdot \log \delta^{-1})$ counters.
  - This is a major improvement over our earlier approach that used $\Theta(\varepsilon^{-1} \cdot \delta^{-1})$ counters.
  - This can be significantly better than just storing a raw frequency count!
- Provides an estimate to within $\varepsilon \|a\|_1$ with probability at least $1 - \delta$. 
Time-Out for Announcements!
Problem Sets

• Solutions to PS3 are now up on the course website.
  • Take a few minutes to read over them – it never hurts to get a different perspective on the solutions to the problems!

• PS4 is due a week from Tuesday. We recommend starting early so you have time to think things over.
Project Checkpoints

• As a reminder, you should be working on the project checkpoint, which is due a week from today.

• Take some time to think through the questions we sent you. Some of them are fairly open-ended and might require you to go looking in the literature for future work. Let us know if you need any help!
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 $\pm 1$ as appropriate.
- To \textit{estimate}(x), return $\text{count}[h(x)]$, multiplied by $\pm 1$ 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 2-independent family of hash functions from \( \mathcal{U} \) to \( w \).

- Choose \( s \in \mathcal{U} \) uniformly randomly and independently of \( h \) from a 2-independent family from \( \mathcal{U} \) to \( \{-1, +1\} \).

- To \textit{increment}(x), add \( s(x) \) to \texttt{count}[h(x)].

- To \textit{estimate}(x), return \( s(x) \cdot \texttt{count}[h(x)] \).
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.
\[ \mathbb{E}[\hat{a}_i] = \mathbb{E}[a_i + \sum_{j \neq i} a_j s(x_i) s(x_j) X_j] \]

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

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

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

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

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

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

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

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

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

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

Since \( s \) is drawn from a 2-independent family of hash functions, we know \( s(x_i) \) and \( s(x_j) \) are independent random variables.
\[ E[\hat{a}_i] = E[a_i + \sum_{j \neq i} a_j s(x_i) s(x_j) X_j] \]
\[ = E[a_i] + E[\sum_{j \neq i} a_j s(x_i) s(x_j) X_j] \]
\[ = a_i + \sum_{j \neq i} E[a_j s(x_i) s(x_j) X_j] \]
\[ = a_i + \sum_{j \neq i} E[s(x_i)] E[s(x_j)] E[a_j X_j] \]
\[ = a_i + \sum_{j \neq i} 0 \]
\[ = a_i \]

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

\( s \) is drawn from a 2-independent family of hash functions.

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

\[ \Pr[s(x_i) = -1] = \frac{1}{2} \quad \Pr[s(x_i) = +1] = \frac{1}{2} \]
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 $c > 0$, we have

\[
\Pr\left[ |X - \mathbb{E}[X]| > c \right] < \frac{\text{Var}[X]}{c^2}.
\]

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

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

= \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]

In general, \text{Var} is \textit{not} a linear operator.

However, if the terms in the sum are \textit{pairwise uncorrelated}, then \text{Var} is linear.

\textbf{Lemma}: The terms in this sum are uncorrelated.

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

\[s(x) = \pm 1, \quad \text{so} \quad s(x)^2 = 1\]
\[
\text{Var}[\hat{a}_i] = \text{Var}[a_i + \sum_{j \neq i} a_j s(x_i)s(x_j)X_j]
\]

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

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

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

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

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

**Useful Fact:** If \( X \) is an indicator, then \( X^2 = X \).
\[
\text{Var}[\hat{a}_i] = \text{Var}[a_i + \sum_{j \neq i} a_j s(x_i) s(x_j) X_j] \\
= \text{Var}\left[\sum_{j \neq i} a_j s(x_i) s(x_j) X_j\right] \\
= \sum_{j \neq i} \text{Var}[a_j s(x_i) s(x_j) X_j] \\
\leq \sum_{j \neq i} E[(a_j s(x_i) s(x_j) X_j)^2] \\
= \sum_{j \neq i} E[a_j^2 s(x_i)^2 s(x_j)^2 X_j^2] \\
= \sum_{j \neq i} a_j^2 E[X_j^2] \\
= \sum_{j \neq i} a_j^2 E[X_j] \\
\leq \frac{1}{w} \sum_{j \neq i} a_j^2 \\
\]

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

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

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

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

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

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

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

\[ = \frac{1}{w} \sum_{j \neq i} a_j^2 \]

I know this might look really dense, but many of these substeps end up being really useful techniques. These ideas generalize, I promise.
Think of \([a_1, a_2, a_3, \ldots]\) as a vector.

What does the following quantity represent?

\[
\sum_j a_j^2
\]

This is the square of the magnitude of the vector!

The magnitude of a vector is called its \textit{L}_2 \textit{norm} and is denoted \(\|a\|_2\).

\[
\|a\|_2 = \sqrt{\sum_j a_j^2}
\]

Therefore, our above sum is \(\|a\|_2^2\).

\[
\text{Var}[\hat{a}_i] = \frac{1}{w} \sum_{j \neq i} a_j^2 \leq \frac{\|a\|_2^2}{w}
\]
Think of \([a_1, a_2, a_3, \ldots]\) as a vector.

What does the following quantity represent?

\[
\sum_j a_j^2
\]

This is the square of the magnitude of the vector!

The magnitude of a vector is called its \(L_2\) norm and is denoted \(\|a\|_2\).

Therefore, our above sum is \(\|a\|_2^2\).

**Great exercise:** Prove that the \(L_2\) norm of a vector is never greater than the \(L_1\) norm.

\[
\text{Var} [\hat{a}_i] = \frac{1}{w} \sum_{j \neq i} a_j^2 \leq \frac{\|a\|_2^2}{w}
\]
Goal: Make an estimator \( \hat{a} \) for some quantity \( a \) where

With probability at least \( 1 - \delta \),

\[
|\hat{a} - a| \leq \varepsilon \cdot \text{size}(\text{input})
\]

for some measure of the size of the input.

\[
\text{Var}[\hat{a}_i] \leq \frac{\|a\|_2^2}{w}
\]
\[
\Pr[|\hat{a}_i - a_i| > \varepsilon \|a\|_2] < \frac{\text{Var}[\hat{a}_i]}{(\varepsilon \|a\|_2)^2}
\]

Chebyshev’s inequality says that

\[
\Pr[|X - \mathbb{E}[X]| > c] < \frac{\text{Var}[X]}{c^2}.
\]
\[
\Pr[|\hat{a}_i - a_i| > \varepsilon \|a\|_2] < \frac{\text{Var}[\hat{a}_i]}{(\varepsilon \|a\|_2)^2} \leq \frac{\|a\|_2^2}{w} \cdot \frac{1}{(\varepsilon \|a\|_2)^2}
\]

\[
\text{Var}[\hat{a}_i] \leq \frac{\|a\|_2^2}{w}
\]
\[
\Pr[|\hat{a}_i - a_i| > \varepsilon \|a\|_2] \\
< \frac{\text{Var}[\hat{a}_i]}{(\varepsilon \|a\|_2)^2} \\
< \frac{\|a\|_2^2}{w} \cdot \frac{1}{(\varepsilon \|a\|_2)^2} \\
= \frac{1}{w \varepsilon^2}
\]
**Goal:** Make an estimator \( \hat{a} \) for some quantity \( a \) where

With probability at least \( 1 - \delta \),

\[
|\hat{a} - a| \leq \varepsilon \cdot \text{size(input)}
\]

for some measure of input size.

\[
\Pr[|\hat{a}_i - a_i| > \varepsilon \|a\|_2] \leq \frac{1}{w \varepsilon^2}
\]

Pick \( w = e \cdot \varepsilon^{-2} \). Then

\[
\Pr[|\hat{a}_i - a_i| > \varepsilon \|a\|_2] \leq e^{-1}.
\]

We now have a single estimator with a not-so-great chance of giving a good estimate.

How do we fix this?
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 \texttt{increment}(x) in the overall structure, we call \texttt{increment}(x) on each of the underlying data structures.
- The probability that at least one of them provides a good estimate is quite high.
- \textbf{Question:} How do you know which one?

\begin{center}
\begin{tabular}{|c|}
\hline
\textbf{Estimator 1:} & 137 \\
\hline
\textbf{Estimator 2:} & 271 \\
\hline
\textbf{Estimator 3:} & 166 \\
\hline
\textbf{Estimator 4:} & 103 \\
\hline
\textbf{Estimator 5:} & 261 \\
\hline
\end{tabular}
\end{center}
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 $e^{-1}$, 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

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

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

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

\[
e^{-k \cdot d} \quad \text{(for some constant } k)\]

- 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
  $$w = \lceil \frac{e}{\varepsilon^2} \rceil \quad d = \Theta(\log \delta^{-1})$$
- 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 $\Theta(w \cdot d)$, 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 $\|a\|_1$.

• **Question to ponder:** Which would you prefer if your elements are more uniform? Which would you prefer if a few elements are extremely common?
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!