Hashing and Sketching
Part Two
Outline for Today

- **Recap from Last Time**
  - Where are we, again?

- **Count Sketches**
  - A frequency estimator that shows off several key mathematical techniques.

- **Cardinality Estimators**
  - How many different items have you seen?
Recap from Last Time
**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.
# How to Build an Estimator

<table>
<thead>
<tr>
<th><strong>Step One:</strong> Build a Simple Estimator</th>
<th><strong>Count-Min Sketch</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Hash items to counters; add +1 when item seen.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Step Two:</strong> Compute Expected Value of Estimator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sum of indicators; 2-independent hashes have low collision rate.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Step Three:</strong> Apply Concentration Inequality</th>
</tr>
</thead>
<tbody>
<tr>
<td>One-sided error; use expected value and Markov’s inequality.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Step Four:</strong> Replicate to Boost Confidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Take min; only fails if all estimates are bad.</td>
</tr>
</tbody>
</table>
New Stuff!
The Count Sketch
Frequency Estimation

- **Recall:** A frequency estimator is a data structure that supports
  - `increment(x)`, which increments the number of times that we’ve seen `x`, and
  - `estimate(x)`, which returns an estimate of how many times we’ve seen `x`.

- **Notation:** Assume that the elements we’re processing are `x_1, ..., x_n`, and that the true frequency of element `x_i` is `a_i`.

- Remember that the frequencies are not random variables – we’re assuming that they’re not under our control. Any randomness comes from hash functions.
# How to Build an Estimator

<table>
<thead>
<tr>
<th>Step One: Build a Simple Estimator</th>
<th>Count-Min Sketch</th>
<th>Count Sketch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hash items to counters; add +1 when item seen.</td>
<td>Sum of indicators; 2-independent hashes have low collision rate.</td>
<td></td>
</tr>
<tr>
<td>Step Two: Compute Expected Value of Estimator</td>
<td>One-sided error; use expected value and Markov’s inequality.</td>
<td></td>
</tr>
<tr>
<td>Step Three: Apply Concentration Inequality</td>
<td>Take min; only fails if all estimates are bad.</td>
<td></td>
</tr>
<tr>
<td>Step Four: Replicate to Boost Confidence</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Revisiting Count-Min

We have a good estimate for ☐, since nothing collides with it.

No matter what we do, we’re not going to get a good estimate for ○ because it collides with a very frequent item (◼).

We have a reasonable estimate for □, since it collides with an uncommon item.

Our estimate for ● is way off because of lots of small collisions.
The Setup

- As before, create an array of counters and assign each item a counter.

- **Key New Step:** For each item $x$, assign $x$ either +1 or -1.
  - To *increment* $(x)$, go to $\text{count}[h(x)]$ and add ±1 as appropriate.
  - To *estimate* $(x)$, return $\text{count}[h(x)]$, multiplied by ±1 as appropriate.

\[
\begin{array}{c|c|c|c|c}
-6 & +6 & +8 & +4 \\
\end{array}
\]
The Setup

We have a good estimate for , since nothing collides with it.

We have a reasonable estimate for because the other collisions mostly offset.

No matter what we do, we’re not going to get a good estimate for because it collides with a very frequent item ( ).

We have a reasonable estimate for , since it collides with an uncommon item.
Formalizing This

- Maintain an array of counters of length $w$.
- Pick $h \in \mathcal{H}$ chosen uniformly at random from a 2-independent family of hash functions from $\mathcal{U}$ to $w$.
- Pick $s \in \mathcal{U}$ uniformly randomly and independently of $h$ from a 2-independent family from $\mathcal{U}$ to $\{-1, +1\}$.
- $\textit{increment}(x)$: $\text{count}[h(x)] += s(x)$.
- $\textit{estimate}(x)$: return $s(x) \cdot \text{count}[h(x)]$.

Which counter? ±1?
# How to Build an Estimator

<table>
<thead>
<tr>
<th>Step One: Build a Simple Estimator</th>
<th>Count-Min Sketch</th>
<th>Count Sketch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hash items to counters; add +1 when item seen.</td>
<td>Hash items to counters; add ±1 when item seen.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step Two: Compute Expected Value of Estimator</th>
<th>Count-Min Sketch</th>
<th>Count Sketch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sum of indicators; 2-independent hashes have low collision rate.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step Three: Apply Concentration Inequality</th>
<th>Count-Min Sketch</th>
<th>Count Sketch</th>
</tr>
</thead>
<tbody>
<tr>
<td>One-sided error; use expected value and Markov’s inequality.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step Four: Replicate to Boost Confidence</th>
<th>Count-Min Sketch</th>
<th>Count Sketch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Take min; only fails if all estimates are bad.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The Expectation, Intuitively

- Focus on any element \( x_i \) whose frequency we’re estimating.
- Think about any element that collides with us.
- With 50% probability, it *increases* our estimate.
- With 50% probability, it *decreases* our estimate.
- **Intuition:** The expected value weights both options equally, so our estimator will be unbiased.
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 estimate $\hat{a}_i$ includes an error term of $s(x_i) \cdot s(x_j) \cdot a_j$
- Why?

Answer at https://pollev.com/cs166spr23
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 estimate \( \hat{a}_i \) includes an error term of
  \[
  s(x_i) \cdot s(x_j) \cdot a_j
  \]
- 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 estimate $\hat{a}_i$ includes an error term of

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

• Why?

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

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.

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

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.
\[
\mathbb{E}[\hat{a}_i] = \mathbb{E}[a_i + \sum_{j \neq i} a_j s(x_i) s(x_j) X_j]
\]
\[
= \mathbb{E}[a_i] + \mathbb{E}\left[ \sum_{j \neq i} a_j s(x_i) s(x_j) X_j \right]
\]
\[
= a_i + \sum_{j \neq i} \mathbb{E}[a_j s(x_i) s(x_j) X_j]
\]
\[
= a_i + \sum_{j \neq i} \mathbb{E}[s(x_i)] \mathbb{E}[s(x_j)] \mathbb{E}[a_j X_j]
\]
\[
= a_i + \sum_{j \neq i} 0
\]

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

---

\[
s \text{ is drawn from a 2-independent family of hash functions.}
\]
\[
s(x_i) \text{ is uniform over } \{-1, +1\}
\]
\[
\Pr[s(x_i) = -1] = \frac{1}{2} \quad \Pr[s(x_i) = +1] = \frac{1}{2}
\]
\[
\mathbb{E}[\hat{a}_i] = \mathbb{E}[a_i + \sum_{j \neq i} a_j s(x_i) s(x_j) X_j]
\]
\[
= \mathbb{E}[a_i] + \mathbb{E}\left[ \sum_{j \neq i} a_j s(x_i) s(x_j) X_j \right]
\]
\[
= a_i + \sum_{j \neq i} \mathbb{E}[a_j s(x_i) s(x_j) X_j]
\]
\[
= a_i + \sum_{j \neq i} \mathbb{E}[s(x_i)] \mathbb{E}[s(x_j)] \mathbb{E}[a_j X_j]
\]
\[
= a_i + \sum_{j \neq i} 0
\]
\[
= a_i
\]

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

$s$ is drawn from a 2-independent family of hash functions.

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

\[
\Pr[s(x_i) = -1] = \frac{1}{2} \quad \Pr[s(x_i) = +1] = \frac{1}{2}
\]
How to Build an Estimator

<table>
<thead>
<tr>
<th>Step</th>
<th>Count-Min Sketch</th>
<th>Count Sketch</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Step One:</strong></td>
<td>Build a Simple Estimator</td>
<td>Build a Simple Estimator</td>
</tr>
<tr>
<td></td>
<td>Hash items to counters; add +1 when item seen.</td>
<td>Hash items to counters; add ±1 when item seen.</td>
</tr>
<tr>
<td><strong>Step Two:</strong></td>
<td>Compute Expected Value of Estimator</td>
<td>Compute Expected Value of Estimator</td>
</tr>
<tr>
<td></td>
<td>Sum of indicators; 2-independent hashes have low collision rate.</td>
<td>2-independence breaks up products; ±1 variables have zero expected value.</td>
</tr>
<tr>
<td><strong>Step Three:</strong></td>
<td>Apply Concentration Inequality</td>
<td>Apply Concentration Inequality</td>
</tr>
<tr>
<td></td>
<td>One-sided error; use expected value and Markov’s inequality.</td>
<td>Take min; only fails if all estimates are bad.</td>
</tr>
<tr>
<td><strong>Step Four:</strong></td>
<td>Replicate to Boost Confidence</td>
<td>Replicate to Boost Confidence</td>
</tr>
</tbody>
</table>
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.

• 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

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

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

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

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

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

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

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

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

**Lemma:** The terms in this sum are pairwise uncorrelated.

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

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

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

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

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

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

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

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

**Useful Fact:** If \( X \) is an indicator, then \( X^2 = X \).

\[
X_j^2 = \begin{cases} 
1 & \text{if } h(x_i) = h(x_j) \\
0 & \text{if } h(x_i) \neq h(x_j) 
\end{cases}
\]
\[
\text{Var}[\hat{a}_i] = \text{Var}[a_i + \sum_{j \neq i} a_j s(x_i) s(x_j) X_j]
\]

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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 \(\|a\|^2_2\).

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

What does the following quantity represent?

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

This is the square of the magnitude of the vector!

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

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

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

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

\[
\text{Var}[\hat{a}_i] \leq \frac{1}{w} \sum_{j \neq i} a_j^2 \leq \frac{\|a\|_2^2}{w}
\]
Where We Stand

• We know that

\[ \text{Var}[\hat{a}_i] \leq \frac{\|a\|^2}{w}. \]

• With the count-min sketch, we bounded the probability that we overestimated by more than \( \varepsilon \|a\|_1 \).

• Since the variance is related to \( \|a\|_2 \), for the count sketch we’ll bound the probability that we are more than \( \varepsilon \|a\|_2 \) from our estimate:

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

Chebyshev’s inequality says that
\[
\Pr\left[ |X - \mathbb{E}[X]| \geq c \right] \leq \frac{\text{Var}[X]}{c^2}.
\]
\[
\Pr[|\hat{a}_i - a_i| > \varepsilon \|a\|_2] \leq \frac{\text{Var}[\hat{a}_i]}{(\varepsilon \|a\|_2)^2} \leq \frac{\|a\|_2^2}{w} \cdot \frac{1}{(\varepsilon \|a\|_2)^2}
\]

\begin{align*}
\text{Var}[\hat{a}_i] & \leq \frac{\|a\|_2^2}{w}
\end{align*}
\[ \Pr[|\hat{a}_i - a_i| > \varepsilon \|a\|_2] \]

\[ \leq \frac{\text{Var}[\hat{a}_i]}{(\varepsilon \|a\|_2)^2} \]

\[ \leq \frac{\|a\|_2^2}{w} \cdot \frac{1}{(\varepsilon \|a\|_2)^2} \]

\[ = \frac{1}{w \varepsilon^2} \]
Taking Stock

• We’ve just shown that

\[ \Pr[|\hat{a}_i - a_i| > \varepsilon \|a\|_2] \leq \frac{1}{4} \]

• This means that we \( w = \Theta(\varepsilon^{-2}) \) in order to get a strong bound.
  • Compare with the count sketch, where we related \( w \) to \( \varepsilon^{-1} \).

• **Idea:** Set \( w = 4 \cdot \varepsilon^{-2} \).
  • Why 4? Because I peeked ahead.
How to Build an Estimator

<table>
<thead>
<tr>
<th>Step One: Build a Simple Estimator</th>
<th>Count-Min Sketch</th>
<th>Count Sketch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hash items to counters; add +1 when item seen.</td>
<td>Hash items to counters; add ±1 when item seen.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step Two: Compute Expected Value of Estimator</th>
<th>Count-Min Sketch</th>
<th>Count Sketch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sum of indicators; 2-independent hashes have low collision rate.</td>
<td>2-independence breaks up products; ±1 variables have zero expected value.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step Three: Apply Concentration Inequality</th>
<th>Count-Min Sketch</th>
<th>Count Sketch</th>
</tr>
</thead>
<tbody>
<tr>
<td>One-sided error; use expected value and Markov’s inequality.</td>
<td>Two-sided error; compute variance and use Chebyshev’s inequality.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step Four: Replicate to Boost Confidence</th>
<th>Count-Min Sketch</th>
<th>Count Sketch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Take min; only fails if all estimates are bad.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Running in Parallel

- Imagine we run \( d \) copies of this estimator and call \( \text{estimate}(x) \) on each of our estimators and get back these estimates.
- We need to give back a single number.
- **Question:** How should we aggregate these numbers into a single estimate?

<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>

Answer at [https://pollev.com/cs166spr23](https://pollev.com/cs166spr23)
Running in Parallel

- Unlike last time, we have a two-sided error, so taking the minimum would be a Very Bad Thing.
- Two reasonable options come to mind:
  - Take the \textit{mean} of the estimates.
  - Take the \textit{median} of the estimates.
- \textbf{Question:} Which should we pick?

\begin{itemize}
  \item Estimator 1: 137
  \item Estimator 2: 271
  \item Estimator 3: 166
  \item Estimator 4: 103
  \item Estimator 5: 261
\end{itemize}
• **Claim:** Taking the mean of multiple estimators does increase our probability of being close to the expected value, but not very quickly.

• **Intuition:** Not all outliers are created equal, and outliers far from the target range skew the estimate.

• **The Math:** Averaging $d$ copies of an estimator decreases the variance by a factor of $d$. (Prove this!) By Chebyshev, that decreases the probability of getting a bad answer by a factor of $d$. We’d like something that decays exponentially in $d$. 
Working With Medians

- **Claim:** If we output the median estimate given by the data structures, we have high probability of giving an acceptably close answer.

- **Intuition:** The only way that the median isn’t in the “good” area is if *at least half* the estimates are in the “bad” area.

- Each individual data structure has a “reasonable” chance to be good, so this is very unlikely.
Working With Medians

• Let $D$ 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 $\frac{1}{4}$, we can upper-bound $D$ with a $\text{Binom}(d, \frac{1}{4})$ variable.

• We want to know $\Pr[D > d / 2]$.

• How can we determine this?
Chernoff Bounds

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

  $$\Pr[X \geq \frac{n}{2}] < e^{-n \cdot z(p)}$$

  where $z(p) = (\frac{1}{2} - p)^2 / 2p$.

*Intuition:* For any fixed value of $p$, this quantity decays exponentially quickly as a function of $n$. It’s extremely unlikely that more than half our estimates will be bad.
Chernoff Bounds

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

\[
\Pr[ X \geq \frac{n}{2} ] < e^{-n \cdot z(p)}
\]

where \( z(p) = (\frac{1}{2} - p)^2 / 2p \).

- In our case, \( D \sim \text{Binom}(d, \frac{1}{4}) \), so we know that

\[
\Pr[ D \geq \frac{d}{2} ] \leq e^{-d \cdot z(1/4)} = e^{-d/8}
\]

- Therefore, choosing \( d = 8 \ln \delta^{-1} \) ensures that

\[
\Pr[|\hat{a}_i - a_i| > \varepsilon \|a\|_2] \leq \Pr[ D \geq \frac{d}{2} ] \leq \delta
\]
## How to Build an Estimator

<table>
<thead>
<tr>
<th>Step One: Build a Simple Estimator</th>
<th>Count-Min Sketch</th>
<th>Count Sketch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hash items to counters; add +1 when item seen.</td>
<td>Hash items to counters; add ±1 when item seen.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step Two: Compute Expected Value of Estimator</th>
<th>Count-Min Sketch</th>
<th>Count Sketch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sum of indicators; 2-independent hashes have low collision rate.</td>
<td>2-independence breaks up products; ±1 variables have zero expected value.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step Three: Apply Concentration Inequality</th>
<th>Count-Min Sketch</th>
<th>Count Sketch</th>
</tr>
</thead>
<tbody>
<tr>
<td>One-sided error; use expected value and Markov’s inequality.</td>
<td>Two-sided error; compute variance and use Chebyshev’s inequality.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step Four: Replicate to Boost Confidence</th>
<th>Count-Min Sketch</th>
<th>Count Sketch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Take min; only fails if all estimates are bad.</td>
<td>Take median; only can fail if half of estimates are wrong; use Chernoff.</td>
<td></td>
</tr>
</tbody>
</table>
### The Count Sketch

**Equation:**

\[ w = \lceil 4 \cdot \varepsilon^{-2} \rceil \]

**Diagram:**

- Sampled uniformly and independently from 2-independent families of hash functions.

**Table:**

<table>
<thead>
<tr>
<th>( h_1 )</th>
<th>( s_1 )</th>
<th>31</th>
<th>41</th>
<th>-59</th>
<th>-26</th>
<th>...</th>
<th>58</th>
</tr>
</thead>
<tbody>
<tr>
<td>( h_2 )</td>
<td>( s_2 )</td>
<td>27</td>
<td>-18</td>
<td>28</td>
<td>-18</td>
<td>...</td>
<td>-45</td>
</tr>
<tr>
<td>( h_3 )</td>
<td>( s_3 )</td>
<td>16</td>
<td>-18</td>
<td>-3</td>
<td>39</td>
<td>...</td>
<td>-75</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( h_d )</td>
<td>( s_d )</td>
<td>69</td>
<td>-31</td>
<td>47</td>
<td>-18</td>
<td>...</td>
<td>59</td>
</tr>
</tbody>
</table>
The Count Sketch

\[ w = [4 \cdot \varepsilon^2] \]

\[
\begin{array}{c|c|c|c|c|c}
 h_1 & s_1 & 31 & 40 & -59 & -26 & \ldots & 58 \\
 h_2 & s_2 & 27 & -18 & 28 & -19 & \ldots & -45 \\
 h_3 & s_3 & 16 & -18 & -3 & 40 & \ldots & -75 \\
 \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
 h_d & s_d & 69 & -31 & 47 & -18 & \ldots & 58 \\
\end{array}
\]

increment(x):
  \[
  \text{for } i = 1 \ldots d:\n  \quad \text{count}[i][h_i(x)] += s_i(x)
  \]
The Count Sketch

\[ w = \lceil 4 \cdot \varepsilon^2 \rceil \]

\[
\begin{array}{c|c|c|c|c|c|c|c|c}
\hline
h_1 & s_1 & 31 & 40 & -59 & -26 & \ldots & 58 \\
\hline
h_2 & s_2 & 27 & -18 & 28 & -19 & \ldots & -45 \\
\hline
h_3 & s_3 & 16 & -18 & -3 & 40 & \ldots & -75 \\
\hline
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
\hline
h_d & s_d & 69 & -31 & 47 & -18 & \ldots & 58 \\
\hline
\end{array}
\]

**increment**\((x)\):

```python
for i = 1 \ldots d:
    count[i][h_i(x)] += s_i(x)
```

**estimate**\((x)\):

```python
options = []
for i = 1 \ldots d:
    options += count[i][h_i(x)] * s_i(x)
return medianOf(options)
```
The Final Analysis

- Here’s a comparison of these two structures.
- **Question to ponder:** When is a count-min sketch better than a count sketch, and vice-versa?

---

**Count-Min Sketch**
- Space: $\Theta(\varepsilon^{-1} \cdot \log \delta^{-1})$
- **increment:** $\Theta(\log \delta^{-1})$
- **estimate:** $\Theta(\log \delta^{-1})$
- Accuracy: within $\varepsilon ||a||_1$.

**Count Sketch**
- Space: $\Theta(\varepsilon^{-2} \cdot \log \delta^{-1})$
- **increment:** $\Theta(\log \delta^{-1})$
- **estimate:** $\Theta(\log \delta^{-1})$
- Accuracy: within $\varepsilon ||a||_2$.
Major Ideas Here

- Concentration inequalities are useful tools for showing the right thing probably happens.
  - For one-sided errors, try Markov’s inequality.
  - For two-sided errors, try Chebyshev’s inequality.
  - To bound the probability that lots of things all go wrong, use Chernoff bounds.
  - For more on different mathematical tools like these, check out this blog post by Scott Aaronson.

- Modest success probability can be amplified by running things in parallel.
  - For one-sided errors, try using the min or max.
  - For two-sided errors, try using the median.

- We can estimate quantities using significantly less space than storing those quantities exactly if we’re okay with approximate answers.
Cardinality Estimation
Cardinality Estimation

• A **cardinality estimator** is a data structure supporting the following operations:
  • **see**(x), which records that x has been seen, and
  • **estimate**(), which returns an estimate of the number of **distinct** values we’ve seen.

• In other words, they estimate the cardinality of the set of all items that have been seen.

• These data structures are widely deployed in practice.
  • Databases use them to select which of many different algorithms to run, based on the number of items to process.
  • Websites use them to estimate how many different people have visited the site in a given time window.
Cardinality Estimation

• As with frequency estimation, we can solve the cardinality estimation problem exactly using hash tables or binary search trees using $\Omega(n)$ space.

• To be useful in large-scale data applications, cardinality estimators need to use significantly less space than this.

• **Question:** How low can we go?
Flipping Coins
Flipping Coins

- Here’s a game: I’m going to flip a coin until I get tails. My score is the number of heads that I flip.

- The probability of flipping $k$ or more consecutive heads is $2^{-k}$, so it’s pretty unlikely that I’m going to flip lots of heads in a row.
Flipping Coins

Suppose I show you the following clip of me playing this game.

Which is more likely?

- I played the game once and got really lucky.
- I played the game 256 times and showed you my best run.

Probability you see this after one game: $\frac{1}{512}$.

Probability this is the best you see after 256 games: approximately 23.3%.
Flipping Coins

- **Intuition:** Play this game multiple times and track the maximum number of heads you get in a row.

- If the maximum number of heads we see is $H$, estimate that we played $2^H$ times.

- **Question:** How good of an estimate is this?
Flipping Coins

• Suppose we play this game $n$ times. What’s the probability we see at least $k$ consecutive heads at least once?

$$\text{Pr[see at least } k \text{ heads in } n \text{ games]} = 1 - \text{Pr[never see } k \text{ heads in } n \text{ games]}$$

$$= 1 - \text{Pr[never see } k \text{ heads in one game]}^n$$

$$= 1 - (1 - 2^{-k})^n$$

• What does this function look like?
Play this game $n$ times. What is the probability that our maximum score is $k$ or more?

**Intuition:** We’re well-concentrated around $k = \log_2 n$.

**Fact:** $E[H] \approx \log_2 n + 0.333$, which requires some nontrivial math to prove.

At $2^{10}$ elements, 63% chance of getting 10 or more in a row.

At $2^{10}$ elements, 6% chance of getting 14 or more in a row.
From Coins to Cardinality

- Ultimately, we’re interested in building a cardinality estimator. How does this help us?

- **Idea:** Hash each item in the data stream, and use each hash as the random source for the coin-flipping game.

- Duplicate items give duplicate hashes, which provide duplicate games, which function as if they never happened.

- If we track the highest score across all these games, we can use that to estimate how many games we played, which is equal to the number of distinct elements we saw.
From Coins to Cardinality

- We need some way of going from hash codes to sequences of coin tosses.
- **Idea:** Treat the hash as a sequence of bits. 0 means heads, 1 means tails.
- Then, count how many 0 bits appear consecutively at the end of the number.
A Simple Estimator

• Keep track of a value $H$, initially zero, that records the maximum number of zero bits seen at the end of a number.

• To see an item:
  • Compute a hash code for that item.
  • Compute the number of trailing zeros.
  • Update $H$ if this is a new record.

• To estimate the number of distinct elements:
  • Return $2^H$. 

\[
\begin{align*}
\text{Hash} & : 1101111010111011 \\
\text{Trailing Zeros} & : TTHTTTTHTHTTTHTT
\end{align*}
\]
A Simple Estimator

• How much space does this single estimator need?
• Assume we have an upper bound $U$ on the maximum cardinality. Our hashes never need more than $\Theta(\log U)$ bits.
• Bits required to write down the position of a bit in that hash: $\Theta(\log \log U)$.
• That is an absolutely tiny amount of space compared to storing the elements!
Improving the Estimator

• The current estimator has a few weaknesses.
  • It always outputs a size that’s a power of two, so we’re likely to be off by a full binary order of magnitude.
  • It tends to skew high, since a single unexpected run of heads pushes the whole total up.
• But we have already seen some techniques for improving estimators:
  • Run lots of copies in parallel to reduce the likelihood of any one of them being bad.
  • Use some creative strategy to combine those individual estimates into one really good one.
• And in fact, folks have done just that.
HyperLogLog

- The *HyperLogLog* estimator uses many independent copies of this estimator to produce a very high-quality estimate.
  - Run $m$ copies of the estimator, using a hash function to distribute items to estimators, so that each copy gets roughly a $1/m$ fraction of the items.
  - Compute the harmonic mean of the estimates to mitigate outliers while smoothing between powers of two.
  - Multiply in a debiasing term to mitigate the skew from both the original estimates and the harmonic mean.
- This estimator is used extensively in practice; with about 768 bytes of memory, it can estimate cardinalities for any real-world data stream to about 3% accuracy.
- It’s widely used in database systems, and many open-source implementations are available.
HyperLogLog

- The analysis of HyperLogLog from the original paper is exceedingly difficult, and I haven’t been able to follow along with all the details.
- Hopefully, this intuitive explanation of how it works is enough for you.
- *(Probably?) Open problem:* Find a significantly simpler and cleaner rigorous analysis of HyperLogLog than the original.
Major Ideas We’ve Seen

- You can build a great estimator by running lots of weak estimators in parallel and aggregating the results.
- Indicator variables and linearity of expectation are powerful tools when analyzing sketches.
- Markov’s and Chebyshev’s inequalities are useful for bounding probabilities involving hashing.
- The Chernoff bound is a great tool for showing it’s unlikely for lots of things to go wrong.
Next Time

- **Cuckoo Hashing**
  - Hashing with worst-case efficient lookups.

- **The Erdős-Rényi Model**
  - Random graph theory revisited.

- **Hypergraph Orientation**
  - A beautiful and surprising theory.