Hashing and Sketching
Part One
Randomized Data Structures

• Randomization is a powerful tool for improving efficiency and solving problems under seemingly impossible constraints.

• Over the next three lectures, we’ll explore a sampler of data structures that give a feel for the breadth of what’s out there.

• You can easily spend an entire academic career just exploring this space; take CS265 for more on randomized algorithms!
Where We’re Going

• **Hashing and Sketching (This Week)**
  
  • Using hash functions to count without counting.

• **Cuckoo Hashing (Next Week)**
  
  • Hashing with worst-case $O(1)$ lookups, along with a splash of random hypergraph theory.
Outline for Today

● **Hash Functions**
  • Understanding our basic building blocks.

● **Frequency Estimation**
  • 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.
Preliminaries: *Hash Functions*
Hashing in Practice

• Hash functions are used extensively in programming and software engineering:
  • They make hash tables possible: think C++ std::hash, Python’s __hash__, or Java’s Object.hashCode().
  • They’re used in cryptography: SHA-256, HMAC, etc.

• **Question:** When we’re in Theoryland, what do we mean when we say “hash function?”
Hashing in Theoryland

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

• The codomain is usually a set of the form

$$[m] = \{0, 1, 2, 3, \ldots, m - 1\}$$

$$h : \mathcal{U} \rightarrow [m]$$
Hashing in Theoryland

- **Intuition:** No matter how clever you are with designing a specific hash function, that hash function isn’t random, and so there will be pathological inputs.
  - You can formalize this with the pigeonhole principle.
- **Idea:** Rather than finding the One True Hash Function, we’ll assume we have a collection of hash functions to pick from, and we’ll choose which one to use randomly.
Families of Hash Functions

- A **family** of hash functions is a set $\mathcal{H}$ of hash functions with the same domain and codomain.
- We can then introduce randomness into our data structures by sampling a random hash function from $\mathcal{H}$.
- **Key Point:** The randomness in our data structures almost always derives from the random choice of hash functions, not from the data.

  *Data is adversarial.*

  *Hash function selection is random.*

- **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.
**Goal:** If we pick $h \in \mathcal{H}$ uniformly at random, then $h$ should distribute elements uniformly randomly.
**Goal:** If we pick \( h \in \mathcal{H} \) uniformly at random, then \( h \) should distribute elements uniformly randomly.
Goal: If we pick $h \in \mathcal{H}$ uniformly at random, then $h$ should distribute elements uniformly randomly.
**Goal:** If we pick \( h \in \mathcal{H} \) uniformly at random, then \( h \) should distribute elements uniformly randomly.
**Goal:** If we pick $h \in \mathcal{H}$ uniformly at random, then $h$ should distribute elements uniformly randomly.

**Problem:** A hash function that distributes $n$ elements uniformly at random over $[m]$ requires $\Omega(n \log m)$ space in the worst case.

**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.

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

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

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

---

Find an “obviously bad” family of hash functions that satisfies the distribution property.

Formulate a hypothesis!
**Distribution Property:** Each element should have an equal probability of being placed in each slot.

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

Find an “obviously bad” family of hash functions that satisfies the distribution property.

Discuss with your neighbors!
**Distribution Property:** Each element should have an equal probability of being placed in each slot.

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

**Problem:** This rule doesn’t guarantee that elements are spread out.
**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 its codomain.
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 its codomain.
**Distribution Property:** Each element should have an equal probability of being placed in each slot.

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

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

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 its codomain.

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 its codomain.

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

**Intuition:**
2-independence means any pair of elements is unlikely to collide.
For any $x \in \mathcal{U}$ and random $h \in \mathcal{H}$, the value of $h(x)$ is uniform over its codomain.

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.

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

**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.

$$\Pr[h(x) = h(y)]$$

**Question:** Where did these elements collide with one another?
For any \( x \in \mathcal{U} \) and random \( h \in \mathcal{H} \), the value of \( h(x) \) is uniform over its codomain.

**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.

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

**Question:** Where did these elements collide with one another?
For any $x \in \mathcal{U}$ and random $h \in \mathcal{H}$, the value of $h(x)$ is uniform over its codomain.

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

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

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

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

**Question:** Where did these elements collide with one another?
For any \( x \in \mathcal{U} \) and random \( h \in \mathcal{H} \), the value of \( h(x) \) is uniform over its codomain.

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

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

Question: Where did these elements collide with one another?
For any $x \in \mathcal{U}$ and random $h \in \mathcal{H}$, the value of $h(x)$ is uniform over its codomain.

**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.

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

**Question:** Where did these elements collide with one another?
For any $x \in \mathcal{U}$ and random $h \in \mathcal{H}$, the value of $h(x)$ is uniform over its codomain.

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

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

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

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

**Intuition:**
2-independence means any pair of elements is unlikely to collide.
For any \( x \in \mathcal{U} \) and random \( h \in \mathcal{H} \), the value of \( h(x) \) is uniform over its codomain.

**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.

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

**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.

\[
\Pr[h(x) = h(y)] = \sum_{i=0}^{m-1} \Pr[h(x) = i \land h(y) = i]
\]
\[
= \sum_{i=0}^{m-1} \Pr[h(x) = i] \cdot \Pr[h(y) = i]
\]
For any \( x \in \mathcal{U} \) and random \( h \in \mathcal{H} \), the value of \( h(x) \) is uniform over its codomain.

**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.

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

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

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

**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 its codomain.

**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.

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

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

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

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.

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

**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.

\[
\Pr[h(x) = h(y)] = \sum_{i=0}^{m-1} \Pr[h(x) = i \land h(y) = i] = \sum_{i=0}^{m-1} \Pr[h(x) = i] \cdot \Pr[h(y) = i] = \sum_{i=0}^{m-1} \frac{1}{m^2} = \frac{1}{m}
\]

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?
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.

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.

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”?

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.
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.

Memory used: 16MB
Let $A$ be the true answer. Let $\hat{A}$ be a random variable denoting our estimate.

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

(True answer)

Memory used: 32MB
Let $A$ be the true answer. Let $\hat{A}$ be a random variable denoting our estimate.
Let $A$ be the true answer. Let $\hat{A}$ be a random variable denoting our estimate.

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.

The more resources we allocate, the better our estimate should be.

Observation 4: A good approximation should be tunable.

Memory used: 256MB

$(True\ answer)$
We have two user-provided 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

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.

What does it mean for an approximation to be “good”?
We have two user-provided values
\[ \varepsilon \in (0, 1] \]
\[ \delta \in (0, 1] \]
where \( \varepsilon \) represents \textit{accuracy} and \( \delta \) represents \textit{confidence}.

\textbf{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 the size of the input.

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

\textit{Probably}
We have two user-provided values
\[ \varepsilon \in (0, 1] \]
\[ \delta \in (0, 1] \]
where \( \varepsilon \) represents *accuracy* and \( \delta \) represents *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

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

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

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

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

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

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

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

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

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

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

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

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.

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

$\varepsilon$ small

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

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

for some measure of the size of the input.

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

\( \varepsilon \) medium

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.

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

$\varepsilon$ large

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

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

for some measure of the size of the input.

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

$\varepsilon$ small

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

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

for some measure of the size of the input.

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

What does it mean for an approximation to be “good”?
Time-Out for Announcements!
PS2 / IA2

- PS1 and IA1 were due today at 3:15PM.
  - Need more time? You can submit during the grace period, which ends tomorrow at 3:15PM.

- PS2 and IA2 go out today. They’re due next Thursday at the start of class.
  - Explore balanced trees, data structure isometries, and the Method of Four Russians!
Final Project Logistics

- We’ve posted information about the CS166 final project to the course website.
- The brief summary:
  - You’ll work in teams of three or four.
  - You’ll pick a data structure and become an expert on it.
  - You’ll put together an explanatory article that guides readers on a magical journey to understanding.
  - You’ll do something “interesting” with the topic, broadly construed.
  - You’ll meet with the course staff for a Q&A session to discuss your writeup, “interesting” component, and the topic at large.
- We hope you have fun with this one – you’ll learn a ton in the process of working through this!
Final Project Logistics

• Your first deliverable is a project proposal, which is due next Thursday at the start of class.
  • Because we need to do topic matchmaking, there is no grace period for the project proposal.

• What you need to do:
  • Select a team of 3 – 4 people.
  • Give us an ordered list of your top four project topics, along with two sources for each topic. (One source per topic must be a research paper.)

• We’ve compiled an extensive list of recommended project topics. It’s available up on the course website.
Back to CS166!
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
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 $x$.
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 \( x \).

<table>
<thead>
<tr>
<th>12</th>
<th>6</th>
<th>4</th>
<th>7</th>
</tr>
</thead>
</table>

![Diagram](https://via.placeholder.com/150)
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 with values 12, 6, 5, and 7]
Our Initial Structure

- We can model “assigning each $x_i$ to a counter” by using hash functions.
- Pick a number of counters $w$ (for “width;” more on that later). We’ll choose the exact value of $w$ later.
- Choose, from a family of 2-independent hash functions $H$, a uniformly-random hash function $h : U \rightarrow [w]$.
- Create an array `count` of $w$ counters, each initially zero.
- To `increment(x)`, increment `count[h(x)]`.
- To `estimate(x)`, return `count[h(x)]`.

```
| 137 | 42  | 166 | ... | 161 |
```
Analyzing our Structure
Some Notation

- Let $x_1, x_2, x_3, \ldots$ denote the list of distinct items whose frequencies are being stored.
- Let $a_1, a_2, a_3, \ldots$ denote the frequencies of those items.
  - e.g. $a_i$ is the true number of times $x_i$ is seen.
- Let $\hat{a}_1, \hat{a}_2, \hat{a}_3, \ldots$ denote the estimate our data structure gives for the frequency of each item.
  - e.g. $\hat{a}_i$ is our estimate for how many times $x_i$ has been seen.
- **Important detail:** the $a_i$ values are not random variables (data are chosen adversarially), while the $\hat{a}_i$ values are random variables (they depend on a randomly-sampled hash function).
Our Goal

• We want to show that, with high probability, our estimate isn’t too far from the correct value.

• Mathematically, we want to look at the expression $\hat{a}_i - a_i$ and show that there is a “high probability” that this is “small enough.”

• We need to pin down what “high probability” and “small enough” mean. To do that, let’s first work out, mathematically, what $\hat{a}_i - a_i$ is.
**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

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

**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 $\text{count}[h(x_i)]$. 
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:

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

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

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

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

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

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

\[ E[X_j] = \]
\begin{align*}
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]
\end{align*}

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

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

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

\[
E[X_j] = 1 \cdot \Pr[h(x_i) = h(x_j)] + 0 \cdot \Pr[h(x_i) \neq h(x_j)]
\]
\[ E[\hat{a}_i - a_i] = E[\sum_{j \neq i} a_j X_j] \]
\[ = \sum_{j \neq i} E[a_j X_j] \]
\[ = \sum_{j \neq i} a_j E[X_j] \]

\[ E[X_j] = 1 \cdot \Pr[h(x_i) = h(x_j)] + 0 \cdot \Pr[h(x_i) \neq h(x_j)] = \Pr[h(x_i) = h(x_j)] \]
\[ E[\hat{a}_i - a_i] = E[\sum_{j \neq i} a_j X_j] \]
\[ = \sum_{j \neq i} E[a_j X_j] \]
\[ = \sum_{j \neq i} a_j E[X_j] \]

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

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

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

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

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

\[ = \Pr[h(x_i) = h(x_j)] \]
\[ E[\hat{a}_i - a_i] = E[\sum_{j \neq i} a_j X_j] \]

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

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

---

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

\[ = \Pr[h(x_i) = h(x_j)] \]

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

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

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

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

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

\[
E[X_j] = 1 \cdot \Pr[h(x_i) = h(x_j)] + 0 \cdot \Pr[h(x_i) \neq h(x_j)] \\
= \Pr[h(x_i) = h(x_j)] \\
= \frac{1}{w}
\]
\[ E[\hat{a}_i - a_i] = E\left[ \sum_{j \neq i} a_j X_j \right] \]
\[ = \sum_{j \neq i} E[a_j X_j] \]
\[ = \sum_{j \neq i} a_j E[X_j] \]
\[ = \sum_{j \neq i} \frac{a_j}{w} \]
\[ \leq \frac{||a||_1}{w} \]

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

$$E[\hat{a}_i - a_i] \leq \frac{\|a\|_1}{w}$$

How do we tune $w$ so we’re likely to fall in this range?
\[ \Pr[\hat{a}_i - a_i > \varepsilon \|a\|_1] \]
\[ \Pr \left[ \hat{a}_i - a_i > \varepsilon \|a\|_1 \right] \]
\[ \Pr[\hat{a}_i - a_i > \varepsilon \|a\|_1] \]

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

\[ \Pr[X \geq c] \leq \frac{\mathbb{E}[X]}{c} . \]
\[
\Pr [ \hat{a}_i - a_i > \epsilon \|a\|_1 ] \leq \frac{E [ \hat{a}_i - a_i ]}{\epsilon \|a\|_1}
\]

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

\[
\Pr [ X \geq c ] \leq \frac{E[X]}{c}.
\]
\[
\Pr \left[ \hat{a}_i - a_i > \varepsilon \|a\|_1 \right] \\
\leq \frac{\mathbb{E} \left[ \hat{a}_i - a_i \right]}{\varepsilon \|a\|_1}
\]
\[
\Pr \left[ \hat{a}_i - a_i > \varepsilon \|a\|_1 \right] \leq \frac{\mathbb{E} \left[ \hat{a}_i - a_i \right]}{\varepsilon \|a\|_1}
\]
\[
\Pr[\hat{a}_i - a_i > \varepsilon \|a\|_1] \\
\leq \frac{E[\hat{a}_i - a_i]}{\varepsilon \|a\|_1} \\
E[\hat{a}_i - a_i] \leq \frac{\|a\|_1}{W}
\]
\[
\Pr \left[ \hat{a}_i - a_i > \varepsilon \| a \|_1 \right] \\
\leq \frac{E \left[ \hat{a}_i - a_i \right]}{\varepsilon \| a \|_1} \\
\leq \frac{\| a \|_1 \cdot 1}{w \varepsilon \| a \|_1}
\]
\[ \Pr [ \hat{a}_i - a_i > \varepsilon \| a \|_1 ] \leq \frac{E [ \hat{a}_i - a_i ]}{\varepsilon \| a \|_1} \leq \frac{\| a \|_1}{w} \cdot \frac{1}{\varepsilon \| a \|_1} \]
\[ \Pr [ \hat{a}_i - a_i > \varepsilon \|a\|_1 ] \leq \frac{E [ \hat{a}_i - a_i ]}{\varepsilon \|a\|_1} \leq \frac{\|a\|_1 \cdot 1}{w \varepsilon \|a\|_1} = \frac{1}{\varepsilon w} \]
**Goal:** Make an estimator \( \hat{a} \) for some quantity \( a \) where

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

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

for some measure of input size.

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

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

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

for some measure of input size.

**Probably**

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

**Correct**

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

\[
\Pr[\hat{a}_i - a_i > \varepsilon \|a\|_1] \leq \delta
\]
**Goal:** Make an estimator $\hat{a}$ for some quantity $a$ where

$$\text{With probability at least } 1 - \delta,$$

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

for some measure of input size.

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

**Initial Idea:**

Pick $w = \varepsilon^{-1} \cdot \delta^{-1}$. Then

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

Suppose we’re counting 1,000 distinct items.
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.

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

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

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

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

**Initial Idea:**

Pick \( w = \varepsilon^{-1} \cdot \delta^{-1} \). Then

Pr[\( \hat{a}_i - a_i > \varepsilon \|a\|_1 \)] \leq \delta

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:** \( 1,000 \cdot \varepsilon^{-1} \).
**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.

**Initial Idea:**

Pick \( w = \varepsilon^{-1} \cdot \delta^{-1} \). Then

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

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 $1 - \delta$,

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

for some measure of input size.

Probably

Approximately

Correct

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

We could choose $w = k \cdot \varepsilon^{-1}$ for any constant $k$ to get a failure probability of at most $k^{-1}$. The choice of $e$ is (mostly) arbitrary.

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

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

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

**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 run \( d \) copies of our data structure in parallel with one another.
- Each row has its hash function sampled uniformly at random from our hash family.
- Each time we \textit{increment} an item, we perform the corresponding \textit{increment} operation on each row.

\[
\begin{align*}
\text{\textit{increment}} & \quad \text{\textit{increment}} \\
\end{align*}
\]

\[
w = \lceil e \cdot \varepsilon^{-1} \rceil
\]

| \( h_1 \) | 31 | 41 | 59 | 26 | 53 | ... | 58 |
| \( h_2 \) | 27 | 18 | 28 | 18 | 28 | ... | 45 |
| \( h_3 \) | 16 | 18 | 3  | 39 | 88 | ... | 75 |

\[
d = ??
\]

\[
h_d
\]

| 69 | 31 | 47 | 18 | 5  | ... | 59 |
Running in Parallel

- Let’s run $d$ copies of our data structure in parallel with one another.
- Each row has its hash function sampled uniformly at random from our hash family.
- Each time we **increment** an item, we perform the corresponding **increment** operation on each row.

\[ w = \lceil e \cdot \varepsilon^{-1} \rceil \]

<table>
<thead>
<tr>
<th>$h_1$</th>
<th>31</th>
<th>41</th>
<th>59</th>
<th>26</th>
<th>53</th>
<th>...</th>
<th>58</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_2$</td>
<td>27</td>
<td>18</td>
<td>28</td>
<td>18</td>
<td>28</td>
<td>...</td>
<td>45</td>
</tr>
<tr>
<td>$h_3$</td>
<td>16</td>
<td>18</td>
<td>3</td>
<td>39</td>
<td>88</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>69</td>
<td>31</td>
<td>47</td>
<td>18</td>
<td>5</td>
<td>...</td>
<td>59</td>
</tr>
</tbody>
</table>
Running in Parallel

- Let’s run \(d\) copies of our data structure in parallel with one another.
- Each row has its hash function sampled uniformly at random from our hash family.
- Each time we **increment** an item, we perform the corresponding **increment** operation on each row.

\[ w = \lceil e \cdot \varepsilon^{-1} \rceil \]

<table>
<thead>
<tr>
<th>(h_1)</th>
<th>32</th>
<th>41</th>
<th>59</th>
<th>26</th>
<th>53</th>
<th>...</th>
<th>58</th>
</tr>
</thead>
<tbody>
<tr>
<td>(h_2)</td>
<td>27</td>
<td>18</td>
<td>29</td>
<td>18</td>
<td>28</td>
<td>...</td>
<td>45</td>
</tr>
<tr>
<td>(h_3)</td>
<td>16</td>
<td>18</td>
<td>3</td>
<td>40</td>
<td>88</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>69</td>
<td>31</td>
<td>47</td>
<td>18</td>
<td>5</td>
<td>...</td>
<td>60</td>
</tr>
</tbody>
</table>
Running in Parallel

• Let’s run \( d \) copies of our data structure in parallel with one another.

• Each row has its hash function sampled uniformly at random from our hash family.

• Each time we \textit{increment} an item, we perform the corresponding \textit{increment} operation on each row.

\[
w = \lceil e \cdot \varepsilon^{-1} \rceil
\]

\[
d = ??
\]

\[
\begin{array}{cccccccc}
\hline
& h_1 & h_2 & h_3 & \cdots & h_{d-1} & h_d \\
\hline
h_1 & 32 & 41 & 59 & 26 & 53 & \cdots & 58 \\
\hline
h_2 & 27 & 18 & 29 & 18 & 28 & \cdots & 45 \\
\hline
h_3 & 16 & 18 & 3 & 40 & 88 & \cdots & 75 \\
\hline
\ldots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
\hline
h_d & 69 & 31 & 47 & 18 & 5 & \cdots & 60 \\
\hline
\end{array}
\]
Running in Parallel

- Imagine we call $\textit{estimate}(x)$ 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>

Formulate a hypothesis!
Running in Parallel

- Imagine we call \textit{estimate}(\chi) on each of our estimators and get back these estimates.
- We need to give back a single number.
- \textbf{Question:} How should we aggregate these numbers into a single estimate?

\begin{itemize}
  \item Estimator 1: 137
  \item Estimator 2: 271
  \item Estimator 3: 166
  \item Estimator 4: 103
  \item Estimator 5: 261
\end{itemize}

Discuss with your neighbors!
Running in Parallel

- Imagine we 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>137</th>
</tr>
</thead>
<tbody>
<tr>
<td>Estimator 2:</td>
<td>271</td>
</tr>
<tr>
<td>Estimator 3:</td>
<td>166</td>
</tr>
<tr>
<td>Estimator 4:</td>
<td>103</td>
</tr>
<tr>
<td>Estimator 5:</td>
<td>261</td>
</tr>
</tbody>
</table>
Running in Parallel

- Imagine we 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>
Running in Parallel

- Imagine we 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>

**Intuition:** The smallest estimate returned has the least "noise," and that’s the best guess for the frequency.
Let $\hat{a}_{ij}$ be the estimate from the $j$th copy of the data structure.

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

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

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

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

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

The only way the minimum estimate is inaccurate is if \textit{every} estimate is inaccurate.

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

Our final estimate is \( \min \{ \hat{a}_{ij} \} \).
Let $\hat{a}_{ij}$ be the estimate from the $j$th copy of the data structure. Our final estimate is $\min \{ \hat{a}_{ij} \}$.
\[
\Pr \left[ \min \left\{ \hat{a}_{ij} \right\} - a_i > \varepsilon \|a\|_1 \right]
\]

= \Pr \left[ \prod_{j=1}^{d} \left( \hat{a}_{ij} - a_i > \varepsilon \|a\|_1 \right) \right]

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

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

\[ = \Pr \left[ \bigwedge_{j=1}^{d} \left( \hat{a}_{ij} - a_i > \varepsilon \|a\|_1 \right) \right] \]

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 \( \min \{ \hat{a}_{ij} \} \).
\[
\Pr \left[ \min \left\{ \hat{a}_{ij} \right\} - a_i > \varepsilon \|a\|_1 \right] = \Pr \left[ \Lambda_{j=1}^d \left( \hat{a}_{ij} - a_i > \varepsilon \|a\|_1 \right) \right] = \prod_{j=1}^d \Pr \left[ \hat{a}_{ij} - a_i > \varepsilon \|a\|_1 \right]
\]

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 \( \min \left\{ \hat{a}_{ij} \right\} \)
\[
\Pr \left[ \min \{ \hat{a}_{ij} \} - a_i > \varepsilon \|a\|_1 \right] \\
= \Pr \left[ \bigwedge_{j=1}^{d} \left( \hat{a}_{ij} - a_i > \varepsilon \|a\|_1 \right) \right] \\
= \prod_{j=1}^{d} \Pr \left[ \hat{a}_{ij} - a_i > \varepsilon \|a\|_1 \right]
\]

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

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

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

Our final estimate is
\[
\min \{ \hat{a}_{ij} \}
\]

\[
\Pr[\hat{a}_i - a_i \geq \varepsilon \| a \|_1] \leq e^{-1}
\]
\[ \text{Let } \hat{a}_{ij} \text{ be the estimate from the } j\text{th copy of the data structure.} \]

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 \( 1 - \delta \),

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

for some measure of input size.

\[
\Pr \left[ \min \{ \hat{a}_{i,j} \} - a_i > \varepsilon \| a \|_1 \right] \leq e^{-d}
\]

**Idea:** Choose \( d = -\ln \delta \).

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

\[
\Pr \left[ \min \{ \hat{a}_{i,j} \} - a_i > \varepsilon \| a \|_1 \right] \leq \delta
\]
The Count-Min Sketch

\[ w = \lceil e \cdot \varepsilon^{-1} \rceil \]

\[
\begin{array}{cccccccc}
  h_1 & 31 & 41 & 59 & 26 & 53 & \ldots & 58 \\
  h_2 & 27 & 18 & 28 & 18 & 28 & \ldots & 45 \\
  h_3 & 16 & 18 & 3 & 39 & 88 & \ldots & 75 \\
  \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
  h_d & 69 & 31 & 47 & 18 & 5 & \ldots & 59 \\
\end{array}
\]

Sampled uniformly and independently from a 2-independent family of hash functions
# The Count-Min Sketch

increment(x):
   for i = 1 ... d:
      count[i][h_i(x)]++

<table>
<thead>
<tr>
<th></th>
<th>h_1</th>
<th>h_2</th>
<th>h_3</th>
<th>...</th>
<th>h_d</th>
</tr>
</thead>
<tbody>
<tr>
<td>h_1</td>
<td>31</td>
<td>41</td>
<td>59</td>
<td>26</td>
<td>53</td>
</tr>
<tr>
<td>h_2</td>
<td>27</td>
<td>18</td>
<td>28</td>
<td>18</td>
<td>28</td>
</tr>
<tr>
<td>h_3</td>
<td>16</td>
<td>18</td>
<td>3</td>
<td>39</td>
<td>88</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>h_d</td>
<td>69</td>
<td>31</td>
<td>47</td>
<td>18</td>
<td>5</td>
</tr>
</tbody>
</table>
The Count-Min Sketch

\[
\begin{array}{cccccccc}
   & h_1 & 31 & 41 & 59 & 26 & 53 & \ldots & 58 \\
   & h_2 & 27 & 18 & 28 & 18 & 28 & \ldots & 45 \\
   & h_3 & 16 & 18 & 3 & 39 & 88 & \ldots & 75 \\
   \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
   & h_d & 69 & 31 & 47 & 18 & 5 & \ldots & 59 \\
\end{array}
\]

**increment**(x):
   for i = 1 \ldots d:
      count[i][h_i(x)]++
The Count-Min Sketch

increment(x):
    for i = 1 ... d:
        count[i][h_i(x)]++
The Count-Min Sketch

\[
\begin{array}{ccccccc}
\text{increment}(x): \\
\text{for } i = 1 \ldots d: \\
\text{count}[i][h_i(x)]++ \\
\end{array}
\]

| \( h_1 \) | 32 | 41 | 59 | 26 | 53 | ... | 58 \\
| \( h_2 \) | 27 | 18 | 28 | 19 | 28 | ... | 45 \\
| \( h_3 \) | 16 | 19 | 3 | 39 | 88 | ... | 75 \\
| ... | ... | ... | ... | ... | ... | ... | \\
| \( h_d \) | 69 | 31 | 47 | 18 | 5 | ... | 60 \\

### The Count-Min Sketch

Increment $x$:

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

Estimate $x$:

```python
result = \infty
for i = 1 \ldots d:
    result = \min(result, count[i][h_i(x)])
return result
```
The Count-Min Sketch

\[
\begin{array}{cccccccc}
& h_1 & & h_2 & & h_3 & & \ldots & \\
32 & 41 & 59 & 26 & 53 & \ldots & 58 \\
27 & 18 & 28 & 19 & 28 & \ldots & 45 \\
16 & 19 & 3 & 39 & 88 & \ldots & 75 \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
69 & 31 & 47 & 18 & 5 & \ldots & 60 \\
\end{array}
\]

**increment**(x):
for i = 1 \ldots d:
    count[i][h_i(x)]++

**estimate**(x):
result = \infty
for i = 1 \ldots d:
    result = \min(result, count[i][h_i(x)])
return result
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.
  - Each individual estimator has $\Theta(\varepsilon^{-1})$ counters, and we run $\Theta(\log \delta^{-1})$ copies in parallel.
- 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 $1 - \delta$. 
Major Ideas From Today

• **2-independent hash families** are useful when we want to keep collisions low.

• A “good” approximation of some quantity should have tunable *confidence* and *accuracy* parameters.

• **Sums of indicator variables** are useful for deriving expected values of estimators.

• **Concentration inequalities** like *Markov’s inequality* are useful for showing estimators don’t stay too much from their expected values.

• Good estimators can be built from multiple parallel copies of weaker estimators.
Next Time

- **Count Sketches**
  - An alternative frequency estimator with different time/space bounds.

- **Cardinality Estimation**
  - Estimating how many different items you’ve seen in a data stream.