$k$-Frequent Items (Heavy-Hitters) Problem: Consider a stream of $n$ items $x_1, \ldots, x_n$ (with possible duplicates). Return any item that appears at least $\frac{n}{k}$ times.
**THE FREQUENT ITEMS PROBLEM**

*k-Frequent Items (Heavy-Hitters) Problem*: Consider a stream of $n$ items $x_1, \ldots, x_n$ (with possible duplicates). Return any item at appears at least $\frac{n}{k}$ times.

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
<th>$x_5$</th>
<th>$x_6$</th>
<th>$x_7$</th>
<th>$x_8$</th>
<th>$x_9$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>12</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>10</td>
<td>3</td>
</tr>
</tbody>
</table>
$k$-Frequent Items (Heavy-Hitters) Problem: Consider a stream of $n$ items $x_1, \ldots, x_n$ (with possible duplicates). Return any item that appears at least $\frac{n}{k}$ times.

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
<th>$x_5$</th>
<th>$x_6$</th>
<th>$x_7$</th>
<th>$x_8$</th>
<th>$x_9$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>12</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>10</td>
<td>3</td>
</tr>
</tbody>
</table>
**k-Frequent Items (Heavy-Hitters) Problem**: Consider a stream of $n$ items $x_1, \ldots, x_n$ (with possible duplicates). Return any item that appears at least $\frac{n}{k}$ times.

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
<th>$x_5$</th>
<th>$x_6$</th>
<th>$x_7$</th>
<th>$x_8$</th>
<th>$x_9$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>12</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>10</td>
<td>3</td>
</tr>
</tbody>
</table>

- What is the maximum number of items that must be returned?
  - a) $n$
  - b) $k$
  - c) $n/k$
  - d) $\log n$
**k-Frequent Items (Heavy-Hitters) Problem:** Consider a stream of \( n \) items \( x_1, \ldots, x_n \) (with possible duplicates). Return any item that appears at least \( \frac{n}{k} \) times.

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( x_3 )</th>
<th>( x_4 )</th>
<th>( x_5 )</th>
<th>( x_6 )</th>
<th>( x_7 )</th>
<th>( x_8 )</th>
<th>( x_9 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>12</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>10</td>
<td>3</td>
</tr>
</tbody>
</table>

- What is the maximum number of items that must be returned?
  
a) \( n \)  
b) \( k \)  
c) \( n/k \)  
d) \( \log n \)
**k-Frequent Items (Heavy-Hitters) Problem**: Consider a stream of \( n \) items \( x_1, \ldots, x_n \) (with possible duplicates). Return any item that appears at least \( \frac{n}{k} \) times.

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( x_3 )</th>
<th>( x_4 )</th>
<th>( x_5 )</th>
<th>( x_6 )</th>
<th>( x_7 )</th>
<th>( x_8 )</th>
<th>( x_9 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>12</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>10</td>
<td>3</td>
</tr>
</tbody>
</table>

- What is the maximum number of items that must be returned?
  
a) \( n \)  
b) \( k \)  
c) \( \frac{n}{k} \)  
d) \( \log n \)

- Trivial with \( O(n) \) space: Store the count for each item and return the one that appears \( \geq \frac{n}{k} \) times.
- Can we do it with less space? I.e., without storing all \( n \) items?
Applications of Frequent Items:

- Finding top/viral items (i.e., products on Amazon, videos watched on Youtube, Google searches, etc.)
- Finding very frequent IP addresses sending requests (to detect DoS attacks/network anomalies).
- 'Iceberg queries' for all items in a database with frequency above some threshold.

Generally want very fast detection, without having to scan through database/logs. That is we want to maintain a running list of frequent items that appear in a stream.
Applications of Frequent Items:

- Finding top/viral items (i.e., products on Amazon, videos watched on Youtube, Google searches, etc.)
Applications of Frequent Items:

- Finding top/viral items (i.e., products on Amazon, videos watched on Youtube, Google searches, etc.)
- Finding very frequent IP addresses sending requests (to detect DoS attacks/network anomalies).
Applications of Frequent Items:

• Finding top/viral items (i.e., products on Amazon, videos watched on Youtube, Google searches, etc.)
• Finding very frequent IP addresses sending requests (to detect DoS attacks/network anomalies).
• ‘Iceberg queries’ for all items in a database with frequency above some threshold.
Applications of Frequent Items:

• Finding top/viral items (i.e., products on Amazon, videos watched on Youtube, Google searches, etc.)
• Finding very frequent IP addresses sending requests (to detect DoS attacks/network anomalies).
• ‘Iceberg queries’ for all items in a database with frequency above some threshold.

Generally want very fast detection, without having to scan through database/logs. That is we want to maintain a running list of frequent items that appear in a stream.
Association rule learning: A very common task in data mining is to identify common associations between different events.
Association rule learning: A very common task in data mining is to identify common associations between different events.
Association rule learning: A very common task in data mining is to identify common associations between different events.
**Association rule learning:** A very common task in data mining is to identify common associations between different events.

- Identified via frequent itemset counting. Find all sets of items that appear many times in the same basket.
**Association rule learning:** A very common task in data mining is to identify common associations between different events.

- Identified via **frequent itemset** counting. Find all sets of items that appear many times in the same basket.
- Frequency of an itemset is known as its support.
Association rule learning: A very common task in data mining is to identify common associations between different events.

- Identified via frequent itemset counting. Find all sets of items that appear many times in the same basket.
- Frequency of an itemset is known as its support.
- A single basket includes many different itemsets, and with many different baskets an efficient approach is critical. E.g., baskets are Twitter users and itemsets are subsets of who they follow.
**Issue:** No algorithm using $o(n)$ space can output just the items with frequency $\geq n/k$. Hard to tell between an item with frequency $n/k$ (should be output) and $n/k - 1$ (should not be output).
**Issue:** No algorithm using $o(n)$ space can output just the items with frequency $\geq n/k$. Hard to tell between an item with frequency $n/k$ (should be output) and $n/k - 1$ (should not be output).

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
<th>$x_5$</th>
<th>$x_6$</th>
<th>...</th>
<th>$x_{n-n/k+1}$</th>
<th>...</th>
<th>$x_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>12</td>
<td>9</td>
<td>27</td>
<td>4</td>
<td>101</td>
<td>...</td>
<td>3</td>
<td>...</td>
<td>3</td>
</tr>
</tbody>
</table>

n/k-1 occurrences
**Issue:** No algorithm using $o(n)$ space can output just the items with frequency $\geq n/k$. Hard to tell between an item with frequency $n/k$ (should be output) and $n/k - 1$ (should not be output).

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
<th>$x_5$</th>
<th>$x_6$</th>
<th>...</th>
<th>$x_{n-n/k+1}$</th>
<th>...</th>
<th>$x_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>12</td>
<td>9</td>
<td>27</td>
<td>4</td>
<td>101</td>
<td></td>
<td>3</td>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>

$n/k-1$ occurrences

($\epsilon, k$)-**Frequent Items Problem:** Consider a stream of $n$ items $x_1, \ldots, x_n$. Return a set $F$ of items, including all items that appear at least $\frac{n}{k}$ times and only items that appear at least $(1 - \epsilon) \cdot \frac{n}{k}$ times.
**Issue:** No algorithm using $o(n)$ space can output just the items with frequency $\geq n/k$. Hard to tell between an item with frequency $n/k$ (should be output) and $n/k - 1$ (should not be output).

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
<th>$x_5$</th>
<th>$x_6$</th>
<th>...</th>
<th>$x_{n-n/k+1}$</th>
<th>...</th>
<th>$x_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>12</td>
<td>9</td>
<td>27</td>
<td>4</td>
<td>101</td>
<td>...</td>
<td>3</td>
<td>...</td>
<td>3</td>
</tr>
</tbody>
</table>

($\epsilon, k$)-Frequent Items Problem: Consider a stream of $n$ items $x_1, \ldots, x_n$. Return a set $F$ of items, including all items that appear at least $\frac{n}{k}$ times and only items that appear at least $(1 - \epsilon) \cdot \frac{n}{k}$ times.

- An example of relaxing to a ‘promise problem’: for items with frequencies in $[(1 - \epsilon) \cdot \frac{n}{k}, \frac{n}{k}]$ no output guarantee.
**Today:** Count-min sketch – a random hashing based method closely related to bloom filters.
Today: Count-min sketch – a random hashing based method closely related to bloom filters.

Let \( A[h(x)] \) estimate \( f(x) \), the frequency of \( x \) in the stream. I.e.,

\[
\{|x_i : x_i = x\}.
\]

random hash function \( h \)

m length array \( A \)

\[
\begin{array}{cccccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]
Today: Count-min sketch – a random hashing based method closely related to bloom filters.

Let \( h(x) \) be a random hash function, and \( f(x) \) be the frequency of \( x \) in the stream. We use \( A[h(x)] \) to estimate \( f(x) \), the frequency of \( x \) in the stream. I.e., \(|\{x_i: x_i = x\}|\).
Today: Count-min sketch – a random hashing based method closely related to bloom filters.

\[ A[h(x)] \] to estimate \( f(x) \), the frequency of \( x \) in the stream. I.e., \( |\{x_i : x_i = x\}| \).
Today: Count-min sketch — a random hashing based method closely related to bloom filters.

Let \( h(x) \) be a random hash function. For each input element \( x \), we can estimate its frequency in the stream \( f(x) \) using a count-min sketch.

The sketch consists of an \( m \) length array \( A \) and multiple hash functions \( h \). Each element is hashed to two positions in the array, and the minimum value at those positions is used as an estimate of the frequency of that element.

For example, consider the following:

- \( h(x_1) = 3 \)
- \( h(x_2) = 5 \)
- \( h(x_3) = 1 \)
- \( h(x_4) = 7 \)

The array \( A \) might look like this:

\[
\begin{array}{cccccccc}
1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0
\end{array}
\]

The minimum value at each position gives us an estimate of the frequency of the corresponding element.
Today: Count-min sketch – a random hashing based method closely related to bloom filters.

\[ A[\mathbf{h}(x)] \] to estimate \( f(x) \), the frequency of \( x \) in the stream. I.e., \( \{|x_i : x_i = x\}| \).
Today: Count-min sketch – a random hashing based method closely related to bloom filters.

random hash function $h$

m length array $A$

$\begin{array}{cccccc}
4 & 2 & 1 & 6 & 20 & 1 \\
\end{array}$
Today: Count-min sketch – a random hashing based method closely related to bloom filters.

Will use $A[h(x)]$ to estimate $f(x)$, the frequency of $x$ in the stream. I.e., $|\{x_i : x_i = x\}|$. 

![Diagram of count-min sketch](image-url)
Use $A[h(x)]$ to estimate $f(x)$.

**Claim 1:** We always have $A[h(x)] \geq f(x)$. Why?

---

$f(x)$: frequency of $x$ in the stream (i.e., number of items equal to $x$).  
$h$: random hash function.  
$m$: size of Count-min sketch array.
COUNT-MIN SKETCH ACCURACY

Use $A[h(x)]$ to estimate $f(x)$.

**Claim 1:** We always have $A[h(x)] \geq f(x)$. Why?

- $A[h(x)]$ counts the number of occurrences of any $y$ with $h(y) = h(x)$, including $x$ itself.

---

$f(x)$: frequency of $x$ in the stream (i.e., number of items equal to $x$). $h$: random hash function. $m$: size of Count-min sketch array.
Use $A[h(x)]$ to estimate $f(x)$.

**Claim 1:** We always have $A[h(x)] \geq f(x)$. Why?

- $A[h(x)]$ counts the number of occurrences of any $y$ with $h(y) = h(x)$, including $x$ itself.
- $A[h(x)] = f(x) + \sum_{y \neq x : h(y) = h(x)} f(y)$.

---

$f(x)$: frequency of $x$ in the stream (i.e., number of items equal to $x$). $h$: random hash function. $m$: size of Count-min sketch array.
COUNT-MIN SKETCH ACCURACY

\[ A[h(x)] = f(x) + \sum_{y \neq x: h(y) = h(x)} f(y) \]

error in frequency estimate

\[ f(x): \text{frequency of } x \text{ in the stream (i.e., number of items equal to } x). \quad h: \text{random hash function. } m: \text{size of Count-min sketch array.} \]
COUNT-MIN SKETCH ACCURACY

\[
A[h(x)] = f(x) + \sum_{y \neq x: h(y) = h(x)} f(y) .
\]

**Expected Error:**

\[
\mathbb{E} \left[ \sum_{y \neq x: h(y) = h(x)} f(y) \right] = \\
\text{error in frequency estimate}
\]

\[f(x): \text{ frequency of } x \text{ in the stream (i.e., number of items equal to } x). \quad h: \text{ random hash function. } m: \text{ size of Count-min sketch array.}\]
COUNT-MIN SKETCH ACCURACY

\[ A[h(x)] = f(x) + \sum_{y \neq x: h(y) = h(x)} f(y). \]

**Expected Error:**

\[
\mathbb{E} \left[ \sum_{y \neq x: h(y) = h(x)} f(y) \right] = \sum_{y \neq x} \Pr(h(y) = h(x)) \cdot f(y)
\]

\( f(x) \): frequency of \( x \) in the stream (i.e., number of items equal to \( x \)). \( h \): random hash function. \( m \): size of Count-min sketch array.
**COUNT-MIN SKETCH ACCURACY**

\[
A[h(x)] = f(x) + \sum_{y \neq x: h(y) = h(x)} f(y).
\]

**Expected Error:**

\[
E \left[ \sum_{y \neq x: h(y) = h(x)} f(y) \right] = \sum_{y \neq x} \Pr(h(y) = h(x)) \cdot f(y)
\]

\[
\leq \sum_{y \neq x} \frac{1}{m} \cdot f(y)
\]

- \( f(x) \): frequency of \( x \) in the stream (i.e., number of items equal to \( x \)).
- \( h \): random hash function.
- \( m \): size of Count-min sketch array.
A[h(x)] = f(x) + \sum_{y \neq x: h(y) = h(x)} f(y).

Expected Error:

\[ \mathbb{E} \left[ \sum_{y \neq x: h(y) = h(x)} f(y) \right] = \sum_{y \neq x} \Pr(h(y) = h(x)) \cdot f(y) \]

\[ \leq \sum_{y \neq x} \frac{1}{m} \cdot f(y) = \frac{1}{m} \cdot (n - f(x)) \leq \frac{n}{m} \]

\[ f(x) \text{: frequency of } x \text{ in the stream (i.e., number of items equal to } x). \quad h: \text{ random hash function. } m: \text{ size of Count-min sketch array.} \]
COUNT-MIN SKETCH ACCURACY

\[ A[h(x)] = f(x) + \sum_{y \neq x: h(y) = h(x)} f(y) \]

**Expected Error:**

\[
\mathbb{E} \left[ \sum_{y \neq x: h(y) = h(x)} f(y) \right] = \sum_{y \neq x} \Pr(h(y) = h(x)) \cdot f(y) \\
\leq \sum_{y \neq x} \frac{1}{m} \cdot f(y) = \frac{1}{m} \cdot (n - f(x)) \leq \frac{n}{m}
\]

What is a bound on probability that the error is \( \geq \frac{2n}{m} \)?

---

**Definitions:**
- \( f(x) \): frequency of \( x \) in the stream (i.e., number of items equal to \( x \)).
- \( h \): random hash function.
- \( m \): size of Count-min sketch array.
\[ A[h(x)] = f(x) + \sum_{y \neq x: h(y) = h(x)} f(y) . \]

**Expected Error:**

\[
\mathbb{E}\left[ \sum_{y \neq x: h(y) = h(x)} f(y) \right] = \sum_{y \neq x} \Pr(h(y) = h(x)) \cdot f(y)
\]

\[
\leq \sum_{y \neq x} \frac{1}{m} \cdot f(y) = \frac{1}{m} \cdot (n - f(x)) \leq \frac{n}{m}
\]

**What is a bound on probability that the error is ≥ \( \frac{2n}{m} \)?**

**Markov’s inequality:**

\[
\Pr \left[ \sum_{y \neq x: h(y) = h(x)} f(y) \geq \frac{2n}{m} \right] \leq \frac{1}{2}.
\]

\( f(x) \): frequency of \( x \) in the stream (i.e., number of items equal to \( x \)). \( h \): random hash function. \( m \): size of Count-min sketch array.
\[ A[h(x)] = f(x) + \sum_{y \neq x: h(y) = h(x)} f(y) \]

**Expected Error:**

\[
\mathbb{E} \left[ \sum_{y \neq x: h(y) = h(x)} f(y) \right] = \sum_{y \neq x} \Pr(h(y) = h(x)) \cdot f(y)
\]

\[
\leq \sum_{y \neq x} \frac{1}{m} \cdot f(y) = \frac{1}{m} \cdot (n - f(x)) \leq \frac{n}{m}
\]

What is a bound on probability that the error is \( \geq \frac{2n}{m} \)?

**Markov’s inequality:**

\[
\Pr \left[ \sum_{y \neq x: h(y) = h(x)} f(y) \geq \frac{2n}{m} \right] \leq \frac{1}{2}.
\]

What property of \( h \) is required to show this bound? a) fully random  
              b) pairwise independent  
              c) 2-universal  
              d) locality sensitive

---

\( f(x) \): frequency of \( x \) in the stream (i.e., number of items equal to \( x \)).  
\( h \): random hash function.  
\( m \): size of Count-min sketch array.
COUNT-MIN SKETCH ACCURACY

\[ A[h(x)] = f(x) + \sum_{y \neq x: h(y) = h(x)} f(y) . \]

**Expected Error:**
\[
\mathbb{E}\left[ \sum_{y \neq x: h(y) = h(x)} f(y) \right] = \sum_{y \neq x} \Pr(h(y) = h(x)) \cdot f(y)
\leq \sum_{y \neq x} \frac{1}{m} \cdot f(y) = \frac{1}{m} \cdot (n - f(x)) \leq \frac{n}{m}
\]

What is a bound on probability that the error is \( \geq \frac{2n}{m} \)?

**Markov’s inequality:** \( \Pr\left[ \sum_{y \neq x: h(y) = h(x)} f(y) \geq \frac{2n}{m} \right] \leq \frac{1}{2} \).

What property of \( h \) is required to show this bound? a) fully random b) pairwise independent c) 2-universal d) locality sensitive

\( f(x) \): frequency of \( x \) in the stream (i.e., number of items equal to \( x \)). \( h \): random hash function. \( m \): size of Count-min sketch array.
Claim: For any $x$, with probability at least $1/2$,

$$f(x) \leq A[h(x)] \leq f(x) + \frac{2n}{m}.$$

\[ f(x) \text{: frequency of } x \text{ in the stream (i.e., number of items equal to } x). \quad h: \text{random hash function. } m: \text{size of Count-min sketch array.} \]
**Claim:** For any $x$, with probability at least $1/2$, 

$$f(x) \leq A[h(x)] \leq f(x) + \frac{2n}{m}.$$ 

To solve the $(\epsilon, k)$-Frequent elements problem, set $m = \frac{2k}{\epsilon}$. 

---

$f(x)$: frequency of $x$ in the stream (i.e., number of items equal to $x$). $h$: random hash function. $m$: size of Count-min sketch array.
Claim: For any $x$, with probability at least $1/2$,

$$f(x) \leq A[h(x)] \leq f(x) + \frac{2n}{m}.$$ 

To solve the $(\epsilon, k)$-Frequent elements problem, set $m = \frac{2k}{\epsilon}$.

How can we improve the success probability?

$f(x)$: frequency of $x$ in the stream (i.e., number of items equal to $x$). $h$: random hash function. $m$: size of Count-min sketch array.
**Claim:** For any $x$, with probability at least $1/2$,

$$f(x) \leq A[h(x)] \leq f(x) + \frac{2n}{m}.$$  

To solve the $(\epsilon, k)$-Frequent elements problem, set $m = \frac{2k}{\epsilon}$.

How can we improve the success probability? **Repetition.**

- $f(x)$: frequency of $x$ in the stream (i.e., number of items equal to $x$).
- $h$: random hash function.
- $m$: size of Count-min sketch array.
Estimate \( f(x) \) with \( \tilde{f}(x) = \min_{i \in [t]} A_i[h_i(x)] \). (count-min sketch)

Why min instead of mean or median? The minimum estimate is always the most accurate since they are all overestimates of the true frequency!
Estimate \( f(x) \) with \( \tilde{f}(x) = \min_{i \in \mathbb{A}} A_i[h_i(x)] \). (count-min sketch)

Why min instead of mean or median?

The minimum estimate is always the most accurate since they are all overestimates of the true frequency!
Count-Min Sketch Accuracy

Estimate $f(x)$ with $\tilde{f}(x) = \min_{i \in [t]} A_i[h_i(x)]$. (count-min sketch)

Why min instead of mean or median?

The minimum estimate is always the most accurate since they are all overestimates of the true frequency!
Estimate $f(x)$ with $\tilde{f}(x) = \min_{i \in [t]} A_i(h_i(x))$. (count-min sketch)

Why min instead of mean or median?
The minimum estimate is always the most accurate since they are all overestimates of the true frequency!
Estimate $f(x)$ with $\tilde{f}(x) = \min_{i \in [t]} A_i[h_i(x)]$. (count-min sketch)
Estimate $f(x)$ with $\tilde{f}(x) = \min_{i \in [t]} A_i[h_i(x)]$. (count-min sketch)
Estimate $f(x)$ with $\tilde{f}(x) = \min_{i \in [t]} A_i[h_i(x)]$. (count-min sketch)

Why min instead of mean or median?
Estimate $f(x)$ with $\tilde{f}(x) = \min_{i \in [t]} A_i[h_i(x)]$. (count-min sketch)

Why min instead of mean or median? The minimum estimate is always the most accurate since they are all overestimates of the true frequency!
**Count-Min Sketch Analysis**

Estimate $f(x)$ by $\tilde{f}(x) = \min_{i \in [t]} A_i[h_i(x)]$

- For every $x$ and $i \in [t]$, we know that for $m = 2^k \epsilon$, with probability $\geq 1/2$:
  
  $f(x) \leq A_i[h_i(x)] \leq f(x) + \epsilon n/k$

- What is $\Pr[f(x) \leq \tilde{f}(x) \leq f(x) + \epsilon n/k]$?

1. $1 - 1/2^t$.

- To get a good estimate with probability $\geq 1 - \delta$, set $t = \log(1/\delta)$. 

---

![Diagram showing count-min sketch analysis](image)

11
Estimate $f(x)$ by $	ilde{f}(x) = \min_{i \in [t]} A_i[h_i(x)]$

- For every $x$ and $i \in [t]$, we know that for $m = \frac{2k}{\epsilon}$, with probability $\geq 1/2$:
  $$f(x) \leq A_i[h_i(x)] \leq f(x) + \frac{\epsilon n}{k}.$$
Estimate $f(x)$ by $\tilde{f}(x) = \min_{i \in [t]} A_i[h_i(x)]$

- For every $x$ and $i \in [t]$, we know that for $m = \frac{2k}{\epsilon}$, with probability $\geq 1/2$:
  $$f(x) \leq A_i[h_i(x)] \leq f(x) + \frac{\epsilon n}{k}.$$

- What is $\Pr[f(x) \leq \tilde{f}(x) \leq f(x) + \frac{\epsilon n}{k}]$?
Estimate $f(x)$ by $\tilde{f}(x) = \min_{i \in [t]} A_i[h_i(x)]$

• For every $x$ and $i \in [t]$, we know that for $m = \frac{2k}{\epsilon}$, with probability $\geq 1/2$:
  $$f(x) \leq A_i[h_i(x)] \leq f(x) + \frac{\epsilon n}{k}.$$ 

• What is $\Pr[f(x) \leq \tilde{f}(x) \leq f(x) + \frac{\epsilon n}{k}]$? $1 - 1/2^t$. 

(count-min sketch analysis)
Estimate $f(x)$ by $	ilde{f}(x) = \min_{i \in [t]} A_i[h_i(x)]$

- For every $x$ and $i \in [t]$, we know that for $m = \frac{2k}{\epsilon}$, with probability $\geq 1/2$:
  
  $$f(x) \leq A_i[h_i(x)] \leq f(x) + \frac{\epsilon n}{k}.$$

- What is $\Pr[f(x) \leq \tilde{f}(x) \leq f(x) + \frac{\epsilon n}{k}]$? $1 - 1/2^t$.

- To get a good estimate with probability $\geq 1 - \delta$, set $t = \log(1/\delta)$. 

COUNT-MIN SKETCH ANALYSIS
**Upshot:** Count-min sketch lets us estimate the frequency of every item in a stream up to error $\frac{en}{k}$ with probability $\geq 1 - \delta$ in $O(\log(1/\delta) \cdot k/\epsilon)$ space.
Upshot: Count-min sketch lets us estimate the frequency of every item in a stream up to error $\frac{\epsilon n}{k}$ with probability $\geq 1 - \delta$ in $O(\log(1/\delta) \cdot k/\epsilon)$ space.

- Accurate enough to solve the $(\epsilon, k)$-Frequent elements problem – distinguish between items with frequency $\frac{n}{k}$ and those with frequency $(1 - \epsilon)\frac{n}{k}$.
**Upshot:** Count-min sketch lets us estimate the frequency of every item in a stream up to error $\frac{\epsilon n}{k}$ with probability $\geq 1 - \delta$ in $O(\log(1/\delta) \cdot k/\epsilon)$ space.

- Accurate enough to solve the $(\epsilon, k)$-Frequent elements problem – distinguish between items with frequency $\frac{n}{k}$ and those with frequency $(1 - \epsilon)\frac{n}{k}$.
- How should we set $\delta$ if we want a good estimate for all items at once, with 99% probability?
Count-min sketch gives an accurate frequency estimate for every item in the stream. But how do we identify the frequent items without having to store/look up the estimated frequency for all elements in the stream?
Count-min sketch gives an accurate frequency estimate for every item in the stream. But how do we identify the frequent items without having to store/look up the estimated frequency for all elements in the stream?

**One approach:**

- When a new item comes in at step $i$, check if its estimated frequency is $\geq i/k$ and store it if so.
- At step $i$ remove any stored items whose estimated frequency drops below $i/k$.
- Store at most $O(k)$ items at once and have all items with frequency $\geq n/k$ stored at the end of the stream.
Questions on Frequent Elements?