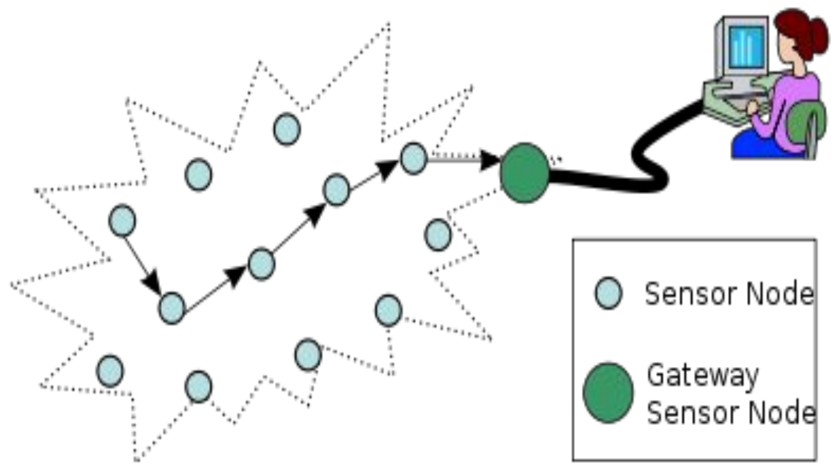
Lecture 7: The Data Stream Model

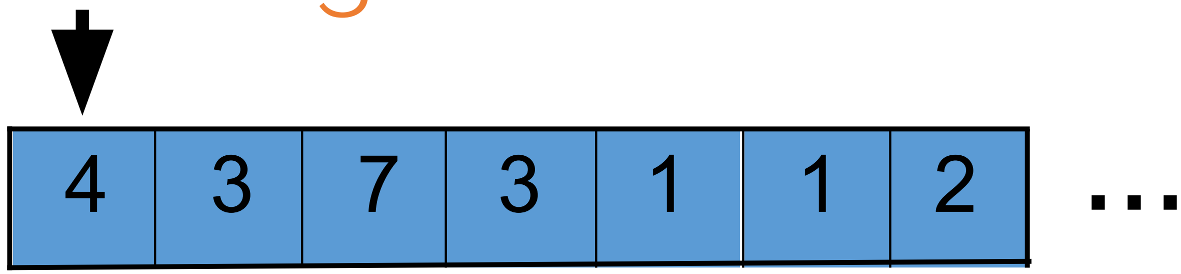
Data Streams

- A stream is a sequence of data, that is too large to be stored in available memory

- Examples
  - Internet search logs
  - Network Traffic
  - Sensor networks
  - Scientific data streams (astronomical, genomics, physical simulations)…



Streaming Model

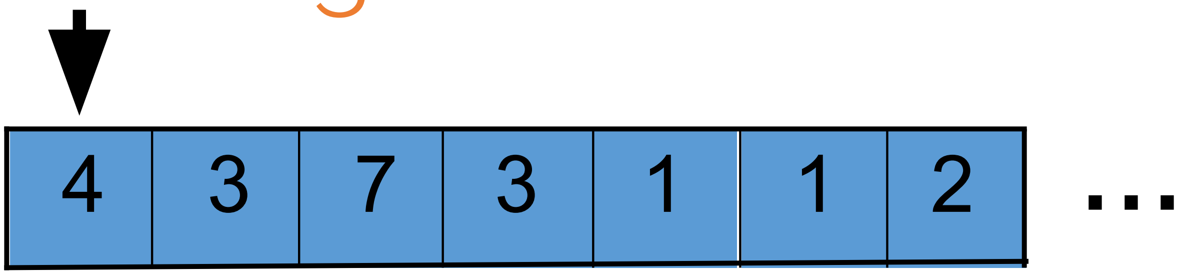


$|\Sigma| = 2^b$

- Stream of elements a_1, …, a_i, … each from an alphabet Σ and taking b bits to represent

- Single or small number of passes over the data

Streaming Model



- Stream of elements $a_1, \ldots, a_i, \ldots$ each from an alphabet Σ and taking b bits to represent
- Single or small number of passes over the data
- Almost all algorithms are randomized and approximate
  - Usually necessary to achieve efficiency
  - Randomness is in the algorithm, not the input
- Goals: minimize space complexity (in bits), processing time

Simple Streaming Problems

- Let $a_{[1:t]} =< a_1, \dots, a_t >$ be the first t elements of the stream

b bits - value
1 bit - sign

- Suppose $a_1, \dots, a_t$ are integers in {$-2^b + 1$, $-2^b + 2$, …, -1, 0, 1, 2, …, 2^b-1}
  - Example stream: 3, 1, 17, 4, -9, 32, 101, 3, -722, 3, 900, 4, 32

Simple Streaming Problems

- Let $a_{[1:t]} = < a_1, \ldots, a_t >$ be the first t elements of the stream
- Suppose $a_1, \ldots, a_t$ are integers in {-$2^b + 1$, -$2^b + 2$, ..., -1, 0, 1, 2, ..., 2^b-1}
  - Example stream: 3, 1, 17, 4, -9, 32, 101, 3, -722, 3, 900, 4, 32
- How many bits do we need to maintain f($a_{[1:t]}$)= $\sum_{i=1,\ldots,t} a_i$?
  - Outputs on example: 3, 4, 21, 25, 16, 48, 149, 152, -570, -567, 333, 337, 379, ...

Simple Streaming Problems

- Let $a_{[1:t]} =< a_1, \ldots, a_t >$ be the first t elements of the stream
- Suppose $a_1, \ldots, a_t$ are integers in {-2^b + 1, -2^b + 2, …, -1, 0, 1, 2, …, 2^b-1}
  - Example stream: 3, 1, 17, 4, -9, 32, 101, 3, -722, 3, 900, 4, 32
- How many bits do we need to maintain $f(a_{[1:t]}) = \sum_{i=1,\ldots,t} a_i$?
  - Outputs on example: 3, 4, 21, 25, 16, 48, 149, 152, -570, -567, 333, 337, 379, …
  - O(b + log t)

$$\underline{(-2^b+1)\cdot t} \le \sum_{i=1\ldots t} a_i \le \underline{(2^b-1)\cdot t}$$

$$O(\log(2^b \cdot t)) = O(b + \log t)$$

Simple Streaming Problems

- Let $a_{[1:t]} =< a_1, \ldots, a_t >$ be the first t elements of the stream
- Suppose $a_1, \ldots, a_t$ are integers in $\{-2^b + 1, -2^b + 2, \ldots, -1, 0, 1, 2, \ldots, 2^b-1\}$
  - Example stream: 3, 1, 17, 4, -9, 32, 101, 3, -722, 3, 900, 4, 32
- How many bits do we need to maintain $f(a_{[1:t]}) = \sum_{i=1,\ldots,t} a_i$?
  - Outputs on example: 3, 4, 21, 25, 16, 48, 149, 152, -570, -567, 333, 337, 379, …
  - O(b + log t)
- How many bits do we need to maintain $f(a_{[1:t]}) = \max_{i=1,\ldots,t} a_i$? $O(b)$
  - Outputs on example: 3, 3, 17, 17, 17, 32, 101, 101, 101, 101, 900, 900, 900, …

Today: Heavy Hitter and Approximate Count

Frequency estimation

Another application of hashing

Applications of Heavy Hitter

- Internet router may want to figure out which IP connections are heavy hitters, e.g., the ones that use more than .01% of your bandwidth

- Or maybe the router wants to know the median (or 90-th percentile) of the file sizes being transferred

Finding ϵ-Heavy Hitters

- S_t is the multiset of items at time t, so $S_0 = \emptyset$, $S_1 = \{a_1\}$, …, $S_i = \{a_1, \ldots, a_i\}$, $\mathrm{count}_t(e) = |\{i \in \{1, 2, \ldots, t\} \text{ such that } a_i = e\}|$
- $e \in \Sigma$ is an ϵ-heavy hitter at time t if $\mathrm{count}_t(e) > \epsilon \cdot t$
- Given $\epsilon > 0$, can we output the ϵ-heavy hitters?
  - Let's output a set of size $\frac{1}{\epsilon}$ containing all the ϵ-heavy hitters

Finding ϵ-Heavy Hitters

- S_t is the multiset of items at time t, so $S_0 = \emptyset, S_1 = \{a_1\}, \ldots, S_i = \{a_1, \ldots, a_i\}$,
$$\text{count}_t(e) = |\{i \in \{1, 2, \ldots, t\} \text{ such that } a_i = e\}|$$

- $e \in \Sigma$ is an ϵ-heavy hitter at time t if $\text{count}_t(e) > \epsilon \cdot t$

- Given $\epsilon > 0$, can we output the ϵ-heavy hitters?
  - Let's output a set of size $\frac{1}{\epsilon}$ containing all the ϵ-heavy hitters

- Note: can output "false positives" but not allowed to output "false negatives", i.e., not allowed to miss any heavy hitter, but could output non-heavy hitters

Finding ϵ-Heavy Hitters

- Example: E, D, B, D, D_5 D, B, A, C, B_{10} B, E, E, E, E_{15}, E (the subscripts are just to help you count)

- At time 5, the element D is the only 1/3-heavy hitter
- At time 11, both B and D are 1/3-heavy hitters
- At time 15, there is no 1/3-heavy hitter
- At time 16, only E is a 1/3-heavy hitter

D appears 3 times by time 5

Can't afford to keep counts of all items, so how to maintain a short summary to output the ϵ-heavy hitters?

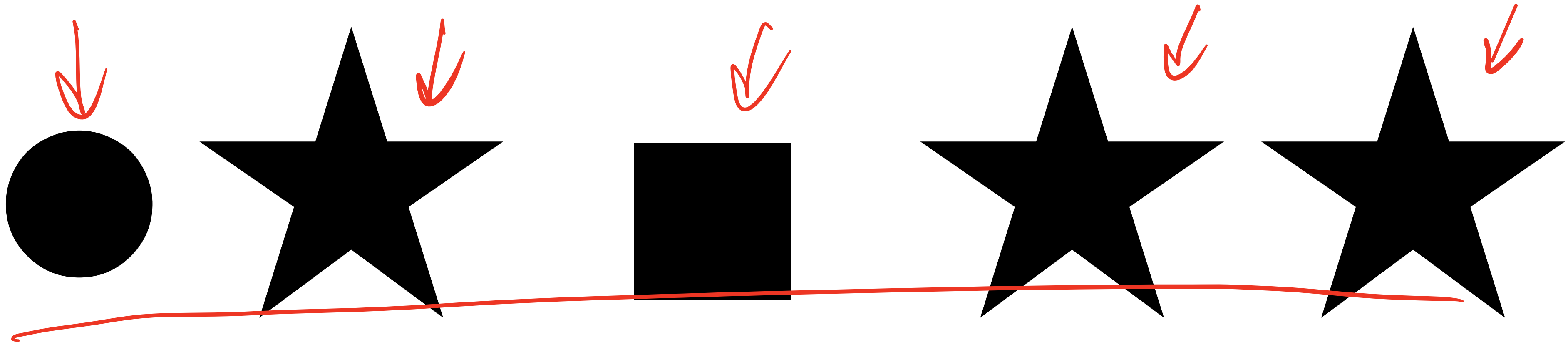
Finding a Majority Element

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

Finding a Majority Element

memory ← empty and counter ← 0
when element a_t arrives
    if (counter == 0)
        memory ← a_t and counter ← 1
    else
        if a_t = memory
            counter + +
        else $a_t \neq$ memory
            counter - -
            (discard a_t)

- At end of the stream, return the element in memory



Memory = _, Count = 0

Memory = _, Count = 1

Memory = _, Count = 0

Memory = _, Count = 1

Memory = _, Count = 0

1

Memory = ●, Count = 1

Memory = ●, Count = 0

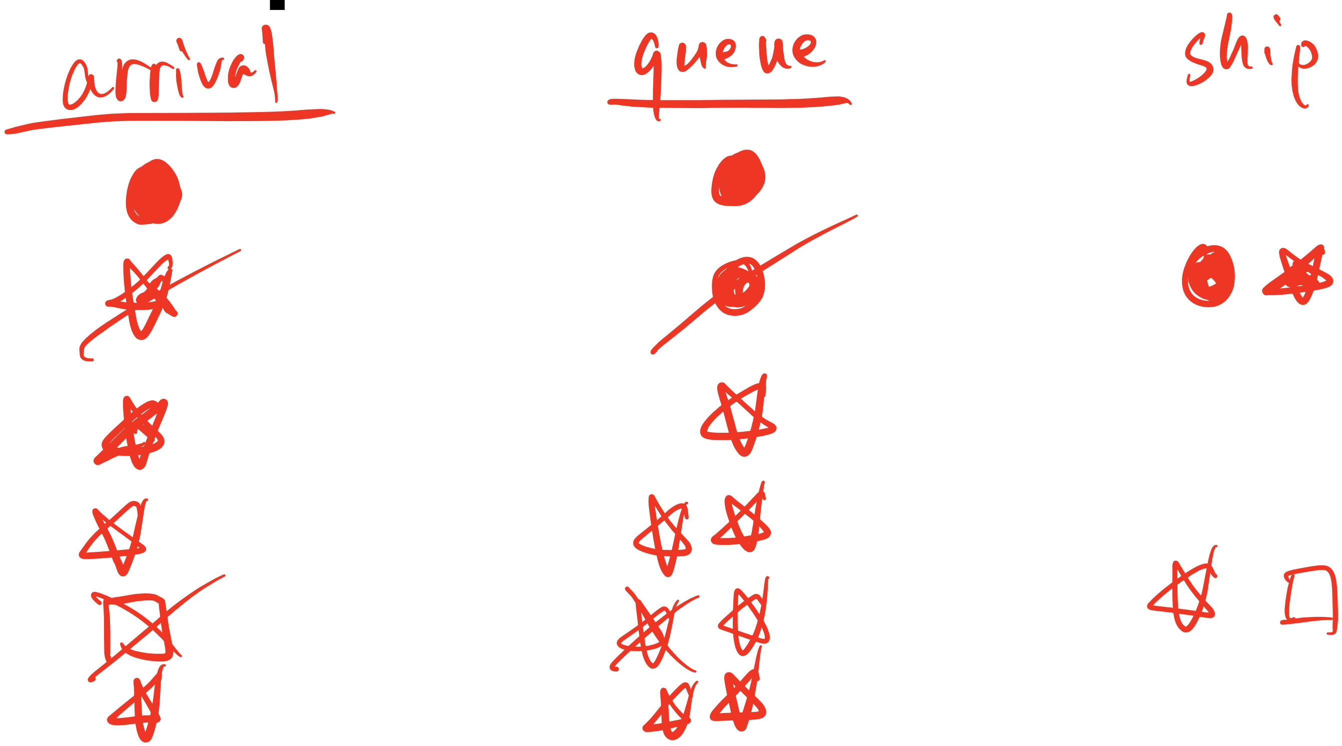
Memory = ■, Count = 1

Memory = ■, Count = 0

Memory = ★, Count = 1

Alternative view of the algorithm

Example: ● ★ ★ ★ ■ ★ ★



Analysis of Finding a Majority Element

- If there is no majority element, we output a false positive, which is OK
- If there is a majority element, we will output it. Why?

If one occurrence of the majority elem is discarded (skipped)
it must be paired with another diff. elem.

b/c the elem is majority, not all copies of it can be paired.

Extending to ε-Heavy Hitters

$\varepsilon = \frac{1}{3}$ $\varepsilon = \frac{1}{10}$

Extending to ε-Heavy Hitters

how many items I keep in memory

Set $\mathrm{k} = \left\lceil \frac{1}{\epsilon} \right\rceil - 1$

Array T[1, …, k], where each location can hold one element from Σ

Array C[1, …, k], where each location can hold a non-negative integer

C[i] ← 0 and $\mathrm{T[i]} \leftarrow \perp$ **for** all i

If there is j ∈ $\{1, 2, \dots, \mathrm{k}\}$ such that $\mathrm{a_t} = \mathrm{T[j]}$, **then** $\mathrm{C[j]}++$

Else if some counter C[j] = 0 **then** $\mathrm{T[j]} \leftarrow \mathrm{a_t}$ and $\mathrm{C[j]} \leftarrow 1$

Else decrement all counters by 1 (and discard element $\mathrm{a_t}$)

$\mathrm{est_t(e)} = \mathrm{C[j]}$ if $\mathrm{e} == \mathrm{T[j]}$ for some j, and $\mathrm{est_t(e)} = 0$ otherwise

Alternative view of the algorithm

Example: ● ★ ◆ ★ ◆ ★ ■ ★ ★ $\varepsilon = 1/3$

arrival	queue	ship
●	● _	
★	● ★	
~~◆~~	~~●~~ ~~★~~	● ★ ◆
★	★ _	
◆	★ ◆	
★	★(2) ◆	

Bounding estimation error

actual count estimated count

$\frac{1}{k+1} = \varepsilon$

- Lemma: $\boxed{0 \le \mathrm{count}_t(e) - \mathrm{est}_t(e)} \le \frac{t}{k+1} \le \epsilon \cdot t$

Proof: $0 \le \mathrm{count}_t(e) - \mathrm{est}_t(e)$ easy

We now prove $\mathrm{count}_t(e) - \mathrm{est}_t(e) \le \frac{t}{k+1} \le \varepsilon t$

observe that $\mathrm{count}_t(e) - \mathrm{est}_t(e)$ = how many copies of e that get discarded (skipped)

whenever e is discarded, it is discarded together with k other diff. elems

Bounding estimation error

- Lemma: $0 \le \text{count}_t(e) - \text{est}_t(e) \le \frac{t}{k+1} \le \epsilon \cdot t$

$\Rightarrow$ e can get discarded at most $\frac{t}{k+1}$ times at time t

Heavy Hitters Guarantee

- At any time t, all ϵ-heavy hitters e are in the array T. Why?

if e is ε-heavy hitter

$$count_t(e) > \varepsilon \cdot t$$

by Lemma (prev slide)

$$est_t(e) > 0$$

Heavy Hitters Guarantee

- At any time t, all ϵ-heavy hitters e are in the array T. Why?

Heavy Hitters Guarantee

- At any time t, all ϵ-heavy hitters e are in the array T. Why?

What is the space requirement? $O\left(\frac{1}{\varepsilon}\left(\log|\Sigma| + \log t\right)\right)$

Heavy Hitters Guarantee

- At any time t, all ϵ-heavy hitters e are in the array T. Why?

What is the space requirement?

- Space is $O(k\ (\log(|\Sigma|) + \log t)) = O(1/\ \epsilon)\ (\log(|\Sigma|) + \log t)$ bits

Frequency Estimation with Deletion

- Suppose we can delete elements e that have already appeared
- Example: (add, A), (add, B), (add, A), (del, B), (del, A), (add, C)

Frequency Estimation with Deletion

- Suppose we can delete elements e that have already appeared
- Example: (add, A), (add, B), (add, A), (del, B), (del, A), (add, C)
- Multisets at different times
  $S_0 = \emptyset, S_1 = \{A\}, S_2 = \{A, B\}, S_3 = \{A, A, \cancel{B}\}, S_4 = \{A, A\}, S_5 = \{A\},$
  $S_6 = \{A, C\}, \ldots$
- "active" set S_t has size $|S_t| = \sum_{e \in \Sigma} \text{count}_t(e)$ and can grow and shrink

99%

Data Structure for Frequency Estimation

$\forall e,$ 0.01

- Query "What is $\text{count}_t(e)$?", should output $\text{est}_t(e)$ with:

$$\Pr[|\text{est}_t(e) - \text{count}_t(e)| \leq \epsilon |S_t|] \geq 1 - \delta$$

- Want space close to our previous O(1/ ϵ) (log($|\Sigma|$) + log t) bits

$\varepsilon \cdot t$

CountMin Sketch: Warmup

- Query "What is $\mathrm{count}_t(e)$?", should output $\mathrm{est}_t(e)$ with:
$$\Pr[|\mathrm{est}_t(e) - \mathrm{count}_t(e)| \leq \epsilon|S_t|] \geq 1 - \delta$$
- Want space close to our previous O(1/ ϵ) (log(|Σ|) + log t) bits
- Let $h: \Sigma \to \{0,1,2, \ldots, k-1\}$ be a hash function (will specify later)
- Maintain an array A[0, 1, …, k-1] to store non-negative integers

when update a_t arrives:

if $a_t = (\mathrm{add}, e)$ **then** $A[h(e)]++$

else $a_t = (\mathrm{del}, e)$, and $A[h(e)]--$

- $\mathrm{est}_t(e) = A[h(e)]$

CountMin: Analyzing the error

- $A[h(e)] = \sum_{e' \in \Sigma} count_t(e') \cdot \boxed{\mathbf{1}(h(e') = h(e))}$, where **1**(condition) evaluates to 1 if the condition is true, and evaluates to 0 otherwise

$$\mathbf{1}(h(e') = h(e)) = \begin{cases} 1 & \text{if } h(e') = h(e) \\ 0 & \text{if } h(e') \neq h(e) \end{cases}$$

CountMin: Analyzing the error

- $A[h(e)] = \sum_{e' \in \Sigma} \text{count}_t(e') \cdot \mathbf{1}(h(e') = h(e))$, where $\mathbf{1}$(condition) evaluates to 1 if the condition is true, and evaluates to 0 otherwise

error

- $A[h(e)] = \underbrace{\text{count}_t(e)} + \underbrace{\sum_{e' \neq e} \text{count}_t(e') \cdot \mathbf{1}(h(e') = h(e))}$,

CountMin: Analyzing the error

- $A[h(e)] = \sum_{e' \in \Sigma} \text{count}_t(e') \cdot \mathbf{1}(h(e') = h(e))$, where **1**(condition) evaluates to 1 if the condition is true, and evaluates to 0 otherwise

- $A[h(e)] = \text{count}_t(e) + \sum_{e' \neq e} \text{count}_t(e') \cdot \mathbf{1}(h(e') = h(e))$,

- $\text{est}_t(e) - \text{count}_t(e) = \sum_{e' \neq e} \text{count}_t(e') \cdot \mathbf{1}(h(e') = h(e))$

CountMin: Analyzing the error

- $A[h(e)] = \sum_{e' \in \Sigma} count_t(e') \cdot \mathbf{1}(h(e') = h(e))$, where $\mathbf{1}$(condition) evaluates to 1 if the condition is true, and evaluates to 0 otherwise

- $A[h(e)]$ = $count_t(e) + \sum_{e' \neq e} count_t(e') \cdot \mathbf{1}(h(e') = h(e))$,

- $est_t(e) - count_t(e) = \sum_{e' \neq e} count_t(e') \cdot \mathbf{1}(h(e') = h(e))$

- Since we have a small array A with k locations, there are likely many $e' \neq e$ with h(e') = h(e), but can we bound the expected error?

CountMin: Analyzing the error

$|A| = k$

- Recall: Family H of hash functions h: U -> {0, 1, …, k-1} is universal if for all $x \neq y$,

$$\Pr_{h \leftarrow H}[h(x) = h(y)] \leq \frac{1}{k}$$

- There is a simple family where h can be specified using O(log |U|) bits. Here, |U| = |Σ|

CountMin: Analyzing the error

- Recall: Family H of hash functions h: U -> {0, 1, …, k-1} is universal if for all $x \neq y$,

$$\Pr_{h \leftarrow H}[h(x) = h(y)] \leq \frac{1}{k}$$

- There is a simple family where h can be specified using O(log |U|) bits. Here, |U| = |Σ|

error

- $E[\underline{\mathrm{est}_t(e) - \mathrm{count}_t(e)}] = E[\underline{\sum_{e' \neq e} \mathrm{count}_t(e') \cdot \mathbf{1}(h(e') = h(e))}]$

$$= \sum_{e' \neq e} \mathrm{count}_t(e') \cdot \underline{E[\mathbb{1}(h(e') = h(e))]}$$

$$= \sum_{e' \neq e} \mathrm{count}_t(e') \cdot \Pr_h[h(e') = h(e)]$$

$$\leq \sum_{e' \neq e} \mathrm{count}_t(e') \cdot \frac{1}{k} \leq |S_t| \cdot \frac{1}{k} = |S_t| \cdot \varepsilon$$

CountMin: Analyzing the error

- Recall: Family H of hash functions h: U -> {0, 1, …, k-1} is universal if for all $x \neq y$,

$$\Pr_{h \leftarrow H}[h(x) = h(y)] \leq \frac{1}{k}$$

- There is a simple family where h can be specified using O(log |U|) bits. Here, |U| = |Σ|

- $E[est_t(e) - count_t(e)] = E[\sum_{e' \neq e} count_t(e') \cdot \mathbf{1}(h(e') = h(e))]$

What is the space requirement?

$k \cdot \log t + \log |\Sigma|$

$= O(\frac{1}{\varepsilon} \cdot \log t + \log |\Sigma|)$

High Probability Bounds for CountMin

Expected error $\leq \frac{|S_t|}{k}$

- Have $0 \leq \text{est}_t(e) - \text{count}_t(e) \leq |S_t|/k$ in expectation from CountMin
  - With probability at least 1/2 $\text{est}_t(e) - \text{count}_t(e) \leq 2|S_t|/k$ Why?

$1-\delta$

High Probability Bounds for CountMin

- Have $0 \leq \mathrm{est}_t(e) - \mathrm{count}_t(e) \leq |S_t|/k$ in expectation from CountMin
  - With probability at least 1/2, $\mathrm{est}_t(e) - \mathrm{count}_t(e) \leq 2|S_t|/k$ Why?
- Can we make the success probability 1-δ?

High Probability Bounds for CountMin

- Have $0 \leq \text{est}_t(e) - \text{count}_t(e) \leq |S_t|/k$ in expectation from CountMin
  - With probability at least 1/2, $\text{est}_t(e) - \text{count}_t(e) \leq 2|S_t|/k$ Why?

- Can we make the success probability 1-δ?
  - Independent repetition: pick m hash functions $h_1, \ldots, h_m$ with $h_i : \Sigma \to \{0, 1, 2, \ldots, k-1\}$ independently from H. Create array A_i for h_i

when update a_t arrives:

for each i from 1 to m

if $a_t = (\text{add}, e)$ **then** $A_i[h_i(e)]++$

else $a_t = (\text{del}, e)$ and $A_i[h_i(e)]--$

High Probability Bounds and Overall Space

What is our new estimate of $\mathrm{count}_t(\mathrm{e})$?

High Probability Bounds and Overall Space

What is our new estimate of $\text{count}_t(e)$?

$$\mathbf{best}_t(e) := \min_{i=1}^{m} A_i[h_i(e)].$$

$\Pr[\text{one copy bad}] \le \frac{1}{2}$

$\Pr[\text{all } m \text{ copies bad}] \le \left(\frac{1}{2}\right)^m \le \delta$

$m = \log\left(\frac{1}{\delta}\right)$

High Probability Bounds and Overall Space

What is our new estimate of $\mathrm{count}_t(e)$?

$$\mathtt{best}_t(e) := \min_{i=1}^{m} A_i[h_i(e)].$$

- Each $A_i[h_i(e)]$ is an *overestimate* to $\mathrm{count}_t(e)$

High Probability Bounds and Overall Space

What is our new estimate of $\mathrm{count}_t(e)$?

$$\mathtt{best}_t(e) := \min_{i=1}^{m} A_i[h_i(e)].$$

- Each $A_i[h_i(e)]$ is an *overestimate* to $\mathrm{count}_t(e)$
- By independence, $\Pr[\text{for all i, } A_i[h_i(e)] - \mathrm{count}_t(e) \geq 2|S_t|/k] \leq \left(\frac{1}{2}\right)^m$

High Probability Bounds and Overall Space

What is our new estimate of $\mathrm{count}_t(e)$?

$$\mathtt{best}_t(e) := \min_{i=1}^{m} A_i[h_i(e)].$$

- Each $A_i[h_i(e)]$ is an *overestimate* to $\mathrm{count}_t(e)$
- By independence, $\Pr[\text{for all i, } A_i[h_i(e)] - \mathrm{count}_t(e) \geq 2|S_t|/k] \leq \left(\frac{1}{2}\right)^m$
- For $k = \frac{2}{\epsilon}$ and $m = \log_2\left(\frac{1}{\delta}\right)$, the error is at most $\epsilon|S_t|$ with probability 1-δ

High Probability Bounds and Overall Space

What is our new estimate of $\mathrm{count}_t(e)$?

$$\mathbf{best}_t(e) := \min_{i=1}^{m} A_i[h_i(e)].$$

- Each $A_i[h_i(e)]$ is an *overestimate* to $\mathrm{count}_t(e)$
- By independence, $\Pr[\text{for all i, } A_i[h_i(e)] - \mathrm{count}_t(e) \geq 2|S_t|/k] \leq \left(\frac{1}{2}\right)^m$
- For $k = \frac{2}{\epsilon}$ and m = $\log_2\left(\frac{1}{\delta}\right)$, the error is at most $\epsilon|S_t|$ with probability 1-δ

What is the space?

CountMin Sketch

- Our new estimate $\mathrm{best}_t(e)$ satisfies

$$\Pr[|\mathrm{best}_t(e) - \mathrm{count}_t(e)| \leq \epsilon |S_t|] \geq 1 - \delta$$

and uses $O(\frac{\log\left(\frac{1}{\delta}\right) \log t}{\epsilon} + \log\left(\frac{1}{\delta}\right) \log |\Sigma|)$ bits of space

CountMin Sketch

- Our new estimate $\text{best}_t(e)$ satisfies

$$\Pr[|\text{best}_t(e) - \text{count}_t(e)| \le \epsilon |S_t|] \ge 1 - \delta$$

  and uses $O(\frac{\log\left(\frac{1}{\delta}\right) \log t}{\epsilon} + \log\left(\frac{1}{\delta}\right) \log |\Sigma|)$ bits of space

- What if we want with probability 9/10, simultaneously for all e, $|\text{best}_t(e) - \text{count}_t(e)| \le \epsilon |S_t|$?

CountMin Sketch

- Our new estimate $\mathrm{best}_t(e)$ satisfies
$$\Pr[|\mathrm{best}_t(e) - \mathrm{count}_t(e)| \leq \epsilon|S_t|] \geq 1 - \delta$$

  and uses $O(\frac{\log\left(\frac{1}{\delta}\right)\log t}{\epsilon} + \log\left(\frac{1}{\delta}\right)\log|\Sigma|)$ bits of space

- What if we want with probability 9/10, simultaneously for all e, $|\mathrm{best}_t(e) - \mathrm{count}_t(e)| \leq \epsilon|S_t|$?

- Set $\delta = \frac{1}{10|\Sigma|}$ and apply a union bound over all $e \in \Sigma$

Other useful streaming algorithms

- Approximate distinct item count
- Random sampling
- Approximate frequency moments