Lecture 6: Streaming

Rafael Oliveira

University of Waterloo
Cheriton School of Computer Science

rafael.oliveira.teaching@gmail.com

May 27, 2021

Overview

- Introduction
  - Data Streaming
  - Basic Examples
- Main Examples
  - Heavy hitters
  - Distinct Elements
  - Weighted Heavy Hitters
- Acknowledgements

Why streaming?

In today's world we have to deal with *big data*. But not all big data are created equal. Today we will study one way in which massive data can appear in our lives: *streaming*.

Why streaming?

In today's world we have to deal with *big data*. But not all big data are created equal. Today we will study one way in which massive data can appear in our lives: *streaming*.

1. Data stream: *massive* sequence of data, too large to store in memory.

Why streaming?

In today's world we have to deal with *big data*. But not all big data are created equal. Today we will study one way in which massive data can appear in our lives: *streaming*.

1. Data stream: *massive* sequence of data, too large to store in memory.
    1. Network traffic (source/destination)
    2. Internet search logs
    3. Database transactions
    4. sensor networks
    5. satellite data feeds

Why streaming?

In today's world we have to deal with *big data*. But not all big data are created equal. Today we will study one way in which massive data can appear in our lives: *streaming*.

1. Data stream: *massive* sequence of data, too large to store in memory.
   1. Network traffic (source/destination)
   2. Internet search logs
   3. Database transactions
   4. sensor networks
   5. satellite data feeds
2. **Does not** come to us at once.

Why streaming?

In today's world we have to deal with *big data*. But not all big data are created equal. Today we will study one way in which massive data can appear in our lives: *streaming*.

1. Data stream: *massive* sequence of data, too large to store in memory.
    1. Network traffic (source/destination)
    2. Internet search logs
    3. Database transactions
    4. sensor networks
    5. satellite data feeds
2. **Does not** come to us at once.
3. Essentially can only look at each piece of data once (or constantly many times)

Why streaming?

In today's world we have to deal with *big data*. But not all big data are created equal. Today we will study one way in which massive data can appear in our lives: *streaming*.

1. Data stream: *massive* sequence of data, too large to store in memory.
   1. Network traffic (source/destination)
   2. Internet search logs
   3. Database transactions
   4. sensor networks
   5. satellite data feeds
2. **Does not** come to us at once.
3. Essentially can only look at each piece of data once (or constantly many times)

How can we deal with it/model it? What can we do if we cannot even see the whole input?

at once

What is streaming?

Definition (Basic Data Stream model)

In the data stream model:

What is streaming?

Definition (Basic Data Stream model)

In the data stream model:

- receive a stream of elements $a_1, a_2, \ldots a_N$ each from a known alphabet Σ. Each element of Σ takes b bits to represent.
    - usually assume that N is known

What is streaming?

Definition (Basic Data Stream model)

In the data stream model:

- receive a stream of elements $a_1, a_2, \ldots a_N$ each from a known alphabet Σ. Each element of Σ takes b bits to represent.
  - usually assume that N is known
- Basic operations (comparison, arithmetic, bitwise) take $\Theta(1)$ time

What is streaming?

Definition (Basic Data Stream model)

In the data stream model:

- receive a stream of elements $a_1, a_2, \ldots a_N$ each from a known alphabet Σ. Each element of Σ takes b bits to represent.
    - usually assume that N is known
- Basic operations (comparison, arithmetic, bitwise) take $\Theta(1)$ time
- Single or small number of passes over data

What is streaming?

Definition (Basic Data Stream model)

In the data stream model:

- receive a stream of elements $a_1, a_2, \ldots a_N$ each from a known alphabet Σ. Each element of Σ takes b bits to represent.
  - usually assume that N is known
- Basic operations (comparison, arithmetic, bitwise) take $\Theta(1)$ time
- Single or small number of passes over data
- Bounded storage
  - Typically $\underbrace{\log^c(N)}_{\text{poly log}(N)}$ for $c = O(1)$ or N^α for some $0 < \alpha < 1$

What is streaming?

Definition (Basic Data Stream model)

In the data stream model:

- receive a stream of elements $a_1, a_2, \ldots a_N$ each from a known alphabet Σ. Each element of Σ takes b bits to represent.
  - usually assume that N is known
- Basic operations (comparison, arithmetic, bitwise) take $\Theta(1)$ time
- Single or small number of passes over data
- Bounded storage
  - Typically $\log^c(N)$ for $c = O(1)$ or N^α for some $0 < \alpha < 1$
- We are allowed to use randomness (almost always necessary)
  - *Probabilistic model:* our algorithm must succeed most of the time

What is streaming?

Definition (Basic Data Stream model)

In the data stream model:

- receive a stream of elements $a_1, a_2, \dots a_N$ each from a known alphabet Σ. Each element of Σ takes b bits to represent.
  - usually assume that N is known
- Basic operations (comparison, arithmetic, bitwise) take $\Theta(1)$ time
- Single or small number of passes over data
- Bounded storage
  - Typically $\log^c(N)$ for $c = O(1)$ or N^α for some $0 < \alpha < 1$
- We are allowed to use randomness (almost always necessary)
  - *Probabilistic model:* our algorithm must succeed most of the time
- (usually) want *approximate answers* to the true answer

What is streaming?

Definition (Basic Data Stream model)

In the data stream model:

- receive a stream of elements $a_1, a_2, \ldots a_N$ each from a known alphabet Σ. Each element of Σ takes b bits to represent.
  - usually assume that N is known
- Basic operations (comparison, arithmetic, bitwise) take $\Theta(1)$ time
- Single or small number of passes over data
- Bounded storage
  - Typically $\log^c(N)$ for $c = O(1)$ or N^α for some $0 < \alpha < 1$
- We are allowed to use randomness (almost always necessary)
  - *Probabilistic model:* our algorithm must succeed most of the time
- (usually) want *approximate answers* to the true answer

Goal: minimize space complexity (in bits) and the processing time.

Examples of Streaming Problems

Example (Sum of elements)

- **Input stream:** $a_1, \ldots, a_N$ be integers from the set $[-2^b + 1, 2^b - 1]$
- **Task:** maintain the current sum of the elements we have seen so far

Examples of Streaming Problems

Example (Sum of elements)

- **Input stream:** $a_1, \ldots, a_N$ be integers from the set $[-2^b + 1, 2^b - 1]$
- **Task:** maintain the current sum of the elements we have seen so far

Example (Median)

- **Input stream:** $a_1, \ldots, a_N$ be integers from the set $[-2^b + 1, 2^b - 1]$
- **Task:** maintain the current median of elements we have seen so far

Examples of Streaming Problems

Example (Distinct elements)

- **Input stream:** $a_1, \ldots, a_N$ be integers from the set $[-2^b + 1, 2^b - 1]$
- **Task:** maintain current $\#$ of distinct elements we have seen so far

Examples of Streaming Problems

Example (Distinct elements)

- **Input stream:** $a_1, \ldots, a_N$ be integers from the set $[-2^b + 1, 2^b - 1]$
- **Task:** maintain current $\#$ of distinct elements we have seen so far

Example (Heavy hitters)

- **Input stream:** $a_1, \ldots, a_N$ integers from $[-2^b + 1, 2^b - 1]$, $\epsilon > 0$
- **Task:** maintain set of elements that contains elements that have appeared at least ϵ-fraction of the time (a.k.a. *heavy hitters*)
- **Constraint:** allowed to also output *false positives* (low hitters), but not allowed to miss any heavy hitter!

Majority Element - Algorithm

Setup: heavy hitters with $\epsilon = 1/2$.

- At time t, we will maintain set S_t which contains the element that has appeared at least $\cancel{n/2}$ $t/2$ times, if any.

Majority Element - Algorithm

Setup: heavy hitters with $\epsilon = 1/2$.

- At time t, we will maintain set S_t which contains the element that has appeared at least $N/2$ times, if any.
- $S_0 = \emptyset$, $c \leftarrow 0$ (c is a counter)

Majority Element - Algorithm

Setup: heavy hitters with $\epsilon = 1/2$.

- At time t, we will maintain set S_t which contains the element that has appeared at least $N/2$ times, if any.
- $S_0 = \emptyset$, $c \leftarrow 0$ (c is a counter)
- when element a_t arrives:

Majority Element - Algorithm

Setup: heavy hitters with $\epsilon = 1/2$.

- At time t, we will maintain set S_t which contains the element that has appeared at least $N/2$ times, if any.
- $S_0 = \emptyset$, $c \leftarrow 0$ (c is a counter)
- when element a_t arrives:
    - If $c == 0$
        - $S_t = \{a_t\}$ and $c \leftarrow 1$

Majority Element - Algorithm

Setup: heavy hitters with $\epsilon = 1/2$.

- At time t, we will maintain set S_t which contains the element that has appeared at least $N/2$ times, if any.
- $S_0 = \emptyset$, $c \leftarrow 0$ (c is a counter)
- when element a_t arrives:
  - If $c == 0$
    - $S_t = \{a_t\}$ and $c \leftarrow 1$
  - Else
    - if $a_t \in S_{t-1}$, set $c \leftarrow c + 1$
    - else $c \leftarrow c - 1$ and discard a_t

Majority Element - Algorithm

Setup: heavy hitters with $\epsilon = 1/2$.

- At time t, we will maintain set S_t which contains the element that has appeared at least $N/2$ times, if any.
- $S_0 = \emptyset$, $c \leftarrow 0$ (c is a counter)
- when element a_t arrives:
  - If $c == 0$
    - $S_t = \{a_t\}$ and $c \leftarrow 1$
  - Else
    - if $a_t \in S_{t-1}$, set $c \leftarrow c + 1$
    - else $c \leftarrow c - 1$ and discard a_t
- At end of stream, return element in S_N

Majority Element - Analysis

- If there is no majority element, we could still output a false positive (low hitter), which is fine.

Example of outputting low hitter:

1 2 1 2 1 2 1 2 ⋯ 1 2 3   no majority element.

$$\underset{0}{\emptyset} \xrightarrow{1} \underset{1}{\{1\}} \xrightarrow{2} \underset{0}{\{1\}} \xrightarrow{1} \underset{1}{\{1\}} \xrightarrow{2} \underset{0}{\{1\}} \rightarrow \cdots \xrightarrow{2} \underset{0}{\{1\}} \xrightarrow{3} \underset{1}{\{3\}}$$

Majority Element - Analysis

- If there is no majority element, we could still output a false positive (low hitter), which is fine.
- What happens when there is a majority element?

Majority Element - Analysis

- If there is no majority element, we could still output a false positive (low hitter), which is fine.
- What happens when there is a majority element?
    - Every time that we discard a copy of the majority element, we throw away a different element.
    - Example: stream $3, 1, 2, 1, 1$

$$\underset{0}{\emptyset} \xrightarrow{3} \underset{1}{\{3\}} \xrightarrow{1} \underset{0}{\{3\}} \xrightarrow{2} \underset{1}{\{2\}} \xrightarrow{1} \underset{0}{\{2\}} \xrightarrow{1} \underset{1}{\{1\}}$$

"throwing away" a 3

threw away 2

Majority Element - Analysis

- If there is no majority element, we could still output a false positive (low hitter), which is fine.
- What happens when there is a majority element?
  - Every time that we discard a copy of the majority element, we throw away a different element.
  - Example: stream $3, 1, 2, 1, 1$
  - Majority element appears more than half the time, so we cannot throw away all the majority elements

Majority Element - Analysis

- If there is no majority element, we could still output a false positive (low hitter), which is fine.
- What happens when there is a majority element?
  - Every time that we discard a copy of the majority element, we throw away a different element.
  - Example: stream $3, 1, 2, 1, 1$
  - Majority element appears more than half the time, so we cannot throw away all the majority elements
- Space used: $\cancel{O(b)}$ (stored set S_t which has at most one element and counter)

$$O(\underbrace{b}_{\text{one element of } \Sigma} + \underbrace{\log N}_{\text{counter}})$$

Heavy hitters Problem

Example (Heavy hitters)

- **Input stream:** $a_1, \ldots, a_N$ integers from $[-2^b + 1, 2^b - 1]$, $\epsilon > 0$
- **Task:** maintain set of elements that contains elements that have appeared at least ϵ-fraction of the time (a.k.a. *heavy hitters*)
- **Constraint:** allowed to also output *false positives* (low hitters), but not allowed to miss any heavy hitter!

Heavy Hitters Algorithm

1. Set $k = \lceil 1/\epsilon \rceil - 1$

Heavy Hitters Algorithm

1. Set $k = \lceil 1/\epsilon \rceil - 1$
2. Set array T of length k where each entry $T[i]$ can hold an element of Σ $(= [-2^b + 1, 2^b - 1])$.

T "array of heavy hitters"

Heavy Hitters Algorithm

1. Set $k = \lceil 1/\epsilon \rceil - 1$
2. Set array T of length k where each entry $T[i]$ can hold an element of Σ $(= [-2^b + 1, 2^b - 1])$.
3. Set array C of length k where each entry can hold non-negative integer

C array of counters for each heavy-hitter

Heavy Hitters Algorithm

1. Set $k = \lceil 1/\epsilon \rceil - 1$
2. Set array T of length k where each entry $T[i]$ can hold an element of Σ $(= [-2^b + 1, 2^b - 1])$.
3. Set array C of length k where each entry can hold non-negative integer
4. Initialize $T[i] \leftarrow NaN$ and $C[i] \leftarrow 0$ for $i \in [k]$.

Heavy Hitters Algorithm

1. Set $k = \lceil 1/\epsilon \rceil - 1$
2. Set array T of length k where each entry $T[i]$ can hold an element of Σ $(= [-2^b + 1, 2^b - 1])$.
3. Set array C of length k where each entry can hold non-negative integer
4. Initialize $T[i] \leftarrow NaN$ and $C[i] \leftarrow 0$ for $i \in [k]$.
5. When receive element a_t:

Heavy Hitters Algorithm

1. Set $k = \lceil 1/\epsilon \rceil - 1$
2. Set array T of length k where each entry $T[i]$ can hold an element of Σ $(= [-2^b + 1, 2^b - 1])$.
3. Set array C of length k where each entry can hold non-negative integer
4. Initialize $T[i] \leftarrow NaN$ and $C[i] \leftarrow 0$ for $i \in [k]$.

5. When receive element a_t:
    1. If there is $j \in [k]$ such that $a_t = T[j]$, then $C[j] \leftarrow C[j] + 1$

if $a_t \in T$ then increase appropriate counter

Heavy Hitters Algorithm

1. Set $k = \lceil 1/\epsilon \rceil - 1$
2. Set array T of length k where each entry $T[i]$ can hold an element of Σ $(= [-2^b + 1, 2^b - 1])$.
3. Set array C of length k where each entry can hold non-negative integer
4. Initialize $T[i] \leftarrow NaN$ and $C[i] \leftarrow 0$ for $i \in [k]$.
5. When receive element a_t:
    1. If there is $j \in [k]$ such that $a_t = T[j]$, then $C[j] \leftarrow C[j] + 1$
    2. Else, if there is $j \in [k]$ such that $C[j] = 0$, then $T[j] \leftarrow a_t$ and $C[j] \leftarrow 1$

in case we have an "empty entry" and $a_t \notin T$

Heavy Hitters Algorithm

1. Set $k = \lceil 1/\epsilon \rceil - 1$
2. Set array T of length k where each entry $T[i]$ can hold an element of Σ $(= [-2^b + 1, 2^b - 1])$.
3. Set array C of length k where each entry can hold non-negative integer
4. Initialize $T[i] \leftarrow NaN$ and $C[i] \leftarrow 0$ for $i \in [k]$.
5. When receive element a_t:
   1. If there is $j \in [k]$ such that $a_t = T[j]$, then $C[j] \leftarrow C[j] + 1$
   2. Else, if there is $j \in [k]$ such that $C[j] = 0$, then $T[j] \leftarrow a_t$ and $C[j] \leftarrow 1$
   3. Else make all $C[j] \leftarrow C[j] - 1$ and discard a_t

array T is full and $a_t \notin T$

Heavy Hitters Algorithm

1. Set $k = \lceil 1/\epsilon \rceil - 1$
2. Set array T of length k where each entry $T[i]$ can hold an element of Σ $(= [-2^b + 1, 2^b - 1])$.
3. Set array C of length k where each entry can hold non-negative integer
4. Initialize $T[i] \leftarrow NaN$ and $C[i] \leftarrow 0$ for $i \in [k]$.
5. When receive element a_t:
   1. If there is $j \in [k]$ such that $a_t = T[j]$, then $C[j] \leftarrow C[j] + 1$
   2. Else, if there is $j \in [k]$ such that $C[j] = 0$, then $T[j] \leftarrow a_t$ and $C[j] \leftarrow 1$
   3. Else make all $C[j] \leftarrow C[j] - 1$ and discard a_t
6. Return the array T with the counter array C

Heavy hitters proof

- For element $e \in \Sigma$, let $est(e) = \begin{cases} C[j], & \text{if } e = T[j] \\ 0, & \text{otherwise.} \end{cases}$

Heavy hitters proof

- For element $e \in \Sigma$, let $est(e) = \begin{cases} C[j], & \text{if } e = T[j] \\ 0, & \text{otherwise.} \end{cases}$

Lemma

Let $count(e)$ be the number of occurrences of e in stream up to time N.

$$0 \leq count(e) - est(e) \leq \frac{N}{k+1} \leq \epsilon N$$

Heavy hitters proof

- For element $e \in \Sigma$, let $est(e) = \begin{cases} C[j], & \text{if } e = T[j] \\ 0, & \text{otherwise.} \end{cases}$

Lemma

Let $count(e)$ be the number of occurrences of e in stream up to time N.

$$0 \leq count(e) - est(e) \leq \frac{N}{k+1} \leq \epsilon N$$

- $count(e) \geq est(e)$ because never increase $C[j]$ for e unless we see e

Heavy hitters proof

- For element $e \in \Sigma$, let $est(e) = \begin{cases} C[j], & \text{if } e = T[j] \\ 0, & \text{otherwise.} \end{cases}$

Lemma

Let $count(e)$ be the number of occurrences of e in stream up to time N.

$$0 \leq count(e) - est(e) \leq \frac{N}{k+1} \leq \epsilon N$$

- $count(e) \geq est(e)$ because never increase $C[j]$ for e unless we see e
- If we don't increase $est(e)$ by 1 when we see an update to e then we decrement k counters and discard current update to e

Heavy hitters proof

- For element $e \in \Sigma$, let $est(e) = \begin{cases} C[j], & \text{if } e = T[j] \\ 0, & \text{otherwise.} \end{cases}$

Lemma

Let $count(e)$ be the number of occurrences of e in stream up to time N.

$$0 \leq count(e) - est(e) \leq \frac{N}{k+1} \leq \epsilon N$$

- $count(e) \geq est(e)$ because never increase $C[j]$ for e unless we see e
- If we don't increase $est(e)$ by 1 when we see an update to e then we decrement k counters and discard current update to e
- So we drop $\underline{k+1 \text{ distinct stream updates}}$, but there are N updates, so we won't increase $est(e)$ by 1 (when we should) at most $\frac{N}{k+1} \leq \epsilon N$ times.

→ elements of T and e

Heavy hitters proof

- At any time N, all heavy hitters e are in T

Heavy hitters proof

- At any time N, all heavy hitters e are in T
  - For an ϵ-heavy hitter e, we have $\underbrace{count(e) > \epsilon \cdot N}_{\text{by definition of being } \epsilon\text{-heavy hitter}}$

Heavy hitters proof

- At any time N, all heavy hitters e are in T
  - For an ϵ-heavy hitter e, we have $count(e) > \epsilon \cdot N$
  - $est(e) \geq count(e) - \epsilon \cdot N > 0$

$count(e) - est(e) \leq \epsilon N$ by Lemma

Heavy hitters proof

- At any time N, all heavy hitters e are in T
  - For an ϵ-heavy hitter e, we have $count(e) > \epsilon \cdot N$
  - $est(e) \geq count(e) - \epsilon \cdot N > 0$
  - $est(e) > 0 \Rightarrow e$ is in T

Heavy hitters proof

- At any time N, all heavy hitters e are in T
  - For an ϵ-heavy hitter e, we have $count(e) > \epsilon \cdot N$
  - $est(e) \geq count(e) - \epsilon \cdot N > 0$
  - $est(e) > 0 \Rightarrow e$ is in T
  - Space used is $O(k \cdot (\underbrace{\log(\Sigma)}_{\text{elements of } \Sigma} + \underbrace{\log N}_{\text{counters}})) = O((1/\epsilon) \cdot (b + \log N))$ bits

- Introduction
  - Data Streaming
  - Basic Examples
- Main Examples
  - Heavy hitters
  - Distinct Elements
  - Weighted Heavy Hitters
- Acknowledgements

Distinct Elements

Example (Distinct elements)

- **Input stream:** $a_1, \ldots, a_N$ be integers from $[0, 2^b - 1]$. $m := 2^b$
- **Task:** maintain current $\#$ of distinct elements D we have seen so far

Distinct Elements

Example (Distinct elements)

- **Input stream:** $a_1, \ldots, a_N$ be integers from $[0, 2^b - 1]$. $m := 2^b$
- **Task:** maintain current $\#$ of distinct elements D we have seen so far

Use strongly 2-universal hash function!

Distinct Elements

Example (Distinct elements)

- **Input stream:** $a_1, \ldots, a_N$ be integers from $[0, 2^b - 1]$. $m := 2^b$
- **Task:** maintain current $\#$ of distinct elements D we have seen so far

Use strongly 2-universal hash function!

- Take strongly 2-universal hash function $h : [0, m-1] \to [0, m^3]$.

Distinct Elements

Example (Distinct elements)

- **Input stream:** $a_1, \ldots, a_N$ be integers from $[0, 2^b - 1]$. $m := 2^b$
- **Task:** maintain current $\#$ of distinct elements D we have seen so far

Use strongly 2-universal hash function!

- Take strongly 2-universal hash function $h : [0, m-1] \to [0, m^3]$.
- From previous lecture, w.h.p. no collisions!

Distinct Elements

Example (Distinct elements)

- **Input stream:** $a_1, \ldots, a_N$ be integers from $[0, 2^b - 1]$. $m := 2^b$
- **Task:** maintain current $\#$ of distinct elements D we have seen so far

Use strongly 2-universal hash function!

- Take strongly 2-universal hash function $h : [0, m-1] \to [0, m^3]$.
- From previous lecture, w.h.p. no collisions!
- Suppose there are D distinct elements $b_1, \ldots, b_D$

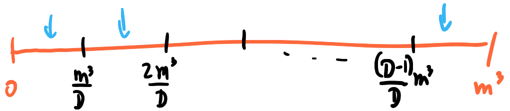
Distinct Elements

Example (Distinct elements)

- **Input stream:** $a_1, \ldots, a_N$ be integers from $[0, 2^b - 1]$. $m := 2^b$
- **Task:** maintain current # of distinct elements *D* we have seen so far

Use strongly 2-universal hash function!

- Take strongly 2-universal hash function $h : [0, m-1] \to [0, m^3]$.
- From previous lecture, w.h.p. no collisions!
- Suppose there are D distinct elements $b_1, \ldots, b_D$
  - If the D hash values $h(b_1), \ldots, h(b_D)$ are *evenly distributed* in $[0, m^3]$, then t^{th} smallest hash value should be close to $\dfrac{tm^3}{D}$.



Distinct Elements

Example (Distinct elements)

- **Input stream:** $a_1, \dots, a_N$ be integers from $[0, 2^b - 1]$. $m := 2^b$
- **Task:** maintain current # of distinct elements D we have seen so far

Use strongly 2-universal hash function!

- Take strongly 2-universal hash function $h : [0, m-1] \to [0, m^3]$.
- From previous lecture, w.h.p. no collisions!
- Suppose there are D distinct elements $b_1, \dots, b_D$
  - If the D hash values $h(b_1), \dots, h(b_D)$ are *evenly distributed* in $[0, m^3]$, then t^{th} smallest hash value should be close to $\frac{tm^3}{D}$.
  - If we know that t^{th} smallest value is T, then $T \approx \frac{tm^3}{D} \Rightarrow D \approx \frac{tm^3}{T}$

can recover D

Distinct Elements - algorithm

- Choose a random hash function h from strongly 2-universal hash family
- For each item a_i in the stream:
  - Compute $h(a_i)$
  - update list that stores the t smallest hash values
  - After all data has read, let T be t^{th} smallest hash value in data stream.

$$\text{Return } Y = \frac{tm^3}{T}$$

Distinct Elements Analysis

- What are our space requirements?

Distinct Elements Analysis

- What are our space requirements?
  - Not going to store the whole hash table, only store hash function h and t numbers (the t smallest values we have seen)

$\underbrace{O(\log m)}_{\text{hash function}}$ $\underbrace{O(t \cdot \log m)}_{t \text{ smallest values}}$

Distinct Elements Analysis

- What are our space requirements?
  - Not going to store the whole hash table, only store hash function h and t numbers (the t smallest values we have seen)
  - Need to find good value of t for have high probability of success

Distinct Elements Analysis

- What are our space requirements?
  - Not going to store the whole hash table, only store hash function h and t numbers (the t smallest values we have seen)
  - Need to find good value of t for have high probability of success

Theorem

Setting $t = O(1/\epsilon^2)$ we have that

$$(1-\epsilon) \cdot D \leq Y \leq (1+\epsilon) \cdot D$$

true # distinct elements

our estimate from algorithm

with constant probability.

Distinct Elements Analysis

Theorem

Setting $t = O(1/\epsilon^2)$ *we have that* $Y = \dfrac{tm^3}{T}$ *satisfies:*

$$(1-\epsilon)\cdot D \leq Y \leq (1+\epsilon)\cdot D$$

with constant probability.

Distinct Elements Analysis

Theorem

Setting $t = O(1/\epsilon^2)$ *we have that* $Y = \dfrac{tm^3}{T}$ *satisfies:*

$$(1-\epsilon)\cdot D \le Y \le (1+\epsilon)\cdot D$$

with constant probability.

Upper Bound: $\underbrace{\Pr[Y > (1+\epsilon)\cdot D]}_{\text{one bad event}}$ "estimate too high"

Distinct Elements Analysis

Theorem

Setting $t = O(1/\epsilon^2)$ we have that $Y = \frac{tm^3}{T}$ satisfies:

$$(1-\epsilon)\cdot D \leq Y \leq (1+\epsilon)\cdot D$$

with constant probability.

Upper Bound: $\Pr[Y > (1+\epsilon)\cdot D]$

- $Y > (1+\epsilon)\cdot D \Rightarrow T < \frac{tm^3}{(1+\epsilon)\cdot D} \leq \frac{(1-\epsilon/2)\cdot tm^3}{D}$

$Y = \frac{tm^3}{T}$

$\frac{1}{1+\epsilon} \leq 1 - \frac{\epsilon}{2}$

Distinct Elements Analysis

Theorem

Setting $t = O(1/\epsilon^2)$ *we have that* $Y = \dfrac{tm^3}{T}$ *satisfies:*

$$(1-\epsilon)\cdot D \le Y \le (1+\epsilon)\cdot D$$

with constant probability.

Upper Bound: $\Pr[Y > (1+\epsilon)\cdot D]$

- $Y > (1+\epsilon)\cdot D \Rightarrow T < \dfrac{tm^3}{(1+\epsilon)\cdot D} \le \dfrac{(1-\epsilon/2)\cdot tm^3}{D}$
- At least t hash values smaller than $\dfrac{(1-\epsilon/2)\cdot tm^3}{D}$

Distinct Elements Analysis

Theorem

Setting $t = O(1/\epsilon^2)$ *we have that* $Y = \frac{tm^3}{T}$ *satisfies:*

$$(1-\epsilon)\cdot D \le Y \le (1+\epsilon)\cdot D$$

with constant probability.

Upper Bound: $\Pr[Y > (1+\epsilon)\cdot D]$

- $Y > (1+\epsilon)\cdot D \Rightarrow T < \frac{tm^3}{(1+\epsilon)\cdot D} \le \frac{(1-\epsilon/2)\cdot tm^3}{D}$
- At least t hash values smaller than $\frac{(1-\epsilon/2)\cdot tm^3}{D}$
- Random variable $X_i = \begin{cases} 1, & \text{if } h(a_i) \le \frac{(1-\epsilon/2)\cdot tm^3}{D} \\ 0, & \text{otherwise} \end{cases}$

Distinct Elements Analysis

Upper Bound: $\Pr[Y > (1+\epsilon) \cdot D]$

- Random variable $X_i = \begin{cases} 1, & \text{if } h(a_i) \leq \dfrac{(1-\epsilon/2) \cdot tm^3}{D} \\ 0, & \text{otherwise} \end{cases}$

Distinct Elements Analysis

Upper Bound: $\Pr[Y > (1+\epsilon)\cdot D]$

- Random variable $X_i = \begin{cases} 1, & \text{if } h(a_i) \le \dfrac{(1-\epsilon/2)\cdot tm^3}{D} \\ 0, & \text{otherwise} \end{cases}$
- $\mathbb{E}[X_i] = \Pr\left[h(a_i) \le \dfrac{(1-\epsilon/2)\cdot tm^3}{D}\right] = \dfrac{(1-\epsilon/2)\cdot t}{D}$

  Each $h(a_i)$ uniformly random in $[0, m^3]$.

Distinct Elements Analysis

Upper Bound: $\Pr[Y > (1+\epsilon)\cdot D]$

- Random variable $X_i = \begin{cases} 1, & \text{if } h(a_i) \leq \dfrac{(1-\epsilon/2)\cdot tm^3}{D} \\ 0, & \text{otherwise} \end{cases}$
- $\mathbb{E}[X_i] = \Pr\left[h(a_i) \leq \dfrac{(1-\epsilon/2)\cdot tm^3}{D}\right] = \dfrac{(1-\epsilon/2)\cdot t}{D}$

  Each $h(a_i)$ uniformly random in $[0, m^3]$.
- If there are D distinct elements,

$$\mathbb{E}\left[\#\text{ elements with hash value } \leq \frac{(1-\epsilon/2)\cdot tm^3}{D}\right] \leq t(1-\epsilon/2)$$

Distinct Elements Analysis

Upper Bound: $\Pr[Y > (1+\epsilon) \cdot D]$

- Random variable $X_i = \begin{cases} 1, & \text{if } h(a_i) \le \dfrac{(1-\epsilon/2) \cdot tm^3}{D} \\ 0, & \text{otherwise} \end{cases}$
- $\mathbb{E}[X_i] = \Pr\left[h(a_i) \le \dfrac{(1-\epsilon/2) \cdot tm^3}{D}\right] = \dfrac{(1-\epsilon/2) \cdot t}{D}$

  Each $h(a_i)$ uniformly random in $[0, m^3]$.
- If there are D distinct elements,

$$\mathbb{E}\left[\#\text{ elements with hash value } \le \frac{(1-\epsilon/2) \cdot tm^3}{D}\right] \le t(1-\epsilon/2)$$

- but we assumed we have at least t such elements! Now need to show that this cannot happen with high probability

have t such elements ">>" expected # elements

Distinct Elements Analysis

Upper Bound: $\Pr[Y > (1+\epsilon) \cdot D]$

- If there are D distinct elements, let $X = \sum_{i=1}^{D} X_i$

$$\mathbb{E}[X] \leq t(1 - \epsilon/2)$$

Distinct Elements Analysis

Upper Bound: $\Pr[Y > (1+\epsilon) \cdot D]$

- If there are D distinct elements, let $X = \sum_{i=1}^{D} X_i$

$$\mathbb{E}[X] \le t(1 - \epsilon/2)$$

Distinct Elements Analysis

Upper Bound: $\Pr[Y > (1+\epsilon) \cdot D]$

- If there are D distinct elements, let $X = \sum_{i=1}^{D} X_i$

$$\mathbb{E}[X] \leq t(1-\epsilon/2)$$

- Probability we will see $\geq t$ elements smaller than $\frac{(1-\epsilon/2) \cdot tm^3}{D}$

Distinct Elements Analysis

Upper Bound: $\Pr[Y > (1+\epsilon) \cdot D]$

- If there are D distinct elements, let $X = \sum_{i=1}^{D} X_i$

$$\mathbb{E}[X] \leq t(1-\epsilon/2)$$

- Probability we will see $\geq t$ elements smaller than $\dfrac{(1-\epsilon/2) \cdot tm^3}{D}$
  - $\text{Var}[X] = \sum_{i=1}^{D} \text{Var}[X_i]$ (pairwise independence)

because of strongly 2-universal hash function

Distinct Elements Analysis

Upper Bound: $\Pr[Y > (1+\epsilon) \cdot D]$

- If there are D distinct elements, let $X = \sum_{i=1}^{D} X_i$

$$\mathbb{E}[X] \le t(1 - \epsilon/2)$$

- Probability we will see $\ge t$ elements smaller than $\dfrac{(1-\epsilon/2)\cdot tm^3}{D}$
  - $\text{Var}[X] = \sum_{i=1}^{D} \text{Var}[X_i]$ (pairwise independence)
  - $\text{Var}[X_i] = \mathbb{E}[(X_i - \mathbb{E}[X_i])^2] = \mathbb{E}[X_i^2] - \mathbb{E}[X_i]^2 \le \mathbb{E}[X_i]$ (indicator variable)

$\mathbb{E}[X_i^2] = \mathbb{E}[X_i]$ because $X_i \sim \{0,1\}$; $\quad \mathbb{E}[X_i]^2 \ge 0$

Distinct Elements Analysis

Upper Bound: $\Pr[Y > (1+\epsilon) \cdot D]$

- If there are D distinct elements, let $X = \sum_{i=1}^{D} X_i$

$$\mathbb{E}[X] \leq t(1 - \epsilon/2)$$

- Probability we will see $\geq t$ elements smaller than $\dfrac{(1-\epsilon/2) \cdot tm^3}{D}$
  - $\mathrm{Var}[X] = \sum_{i=1}^{D} \mathrm{Var}[X_i]$ (pairwise independence)
  - $\mathrm{Var}[X_i] = \mathbb{E}[(X_i - \mathbb{E}[X_i])^2] = \mathbb{E}[X_i^2] - \mathbb{E}[X_i]^2 \leq \mathbb{E}[X_i]$ (indicator variable) $\Rightarrow \mathrm{Var}[X] \leq \mathbb{E}[X]$
  - Chebyshev's inequality:

Chebyshev

$$\Pr[X > t] = \Pr[X > \overbrace{t \cdot (1-\epsilon/2)}^{\mathbb{E}[X]} + \epsilon \cdot t/2]$$

$$\leq \Pr[|X - \mathbb{E}[X]| > \epsilon \cdot t/2] \leq \frac{4 \cdot \mathrm{Var}[X]}{\epsilon^2 t^2} \leq \frac{4}{\epsilon^2 t}$$

because absolute value is bigger event

Distinct Elements Analysis

Lower Bound: $\Pr[Y < (1-\epsilon)\cdot D]$.

Similar calculation as previous slide.[1]
Practice problem: do this part of the proof.

[1] replacing $1-\epsilon$ by $1+\epsilon$ and using Chebyshev

Distinct Elements Analysis

Lower Bound: $\Pr[Y < (1-\epsilon)\cdot D]$.

Similar calculation as previous slide.[1]
Practice problem: do this part of the proof.

- $\Pr[Y > (1+\epsilon)\cdot D] \leq \dfrac{4}{\epsilon^2 t}$
- $\Pr[Y < (1-\epsilon)\cdot D] \leq \dfrac{4}{\epsilon^2 t}$

[1] replacing $1-\epsilon$ by $1+\epsilon$ and using Chebyshev

Distinct Elements Analysis

Lower Bound: $\Pr[Y < (1-\epsilon)\cdot D]$.

Similar calculation as previous slide.[1]
Practice problem: do this part of the proof.

- $\Pr[Y > (1+\epsilon)\cdot D] \leq \dfrac{4}{\epsilon^2 t}$
- $\Pr[Y < (1-\epsilon)\cdot D] \leq \dfrac{4}{\epsilon^2 t}$
- Setting $t = 24/\epsilon^2$ gives us

$$\Pr[(1-\epsilon)\cdot D \leq Y \leq (1+\epsilon)\cdot D] \geq 1 - \frac{8}{\epsilon^2 t} = 2/3$$

$$1 - \Pr[Y > (1+\epsilon)D] - \Pr[Y < (1-\epsilon)D]$$

[1] replacing $1-\epsilon$ by $1+\epsilon$ and using Chebyshev

Distinct Elements Analysis

Lower Bound: $\Pr[Y < (1-\epsilon)\cdot D]$.

Similar calculation as previous slide.[1]
Practice problem: do this part of the proof.

- $\Pr[Y > (1+\epsilon)\cdot D] \le \dfrac{4}{\epsilon^2 t}$
- $\Pr[Y < (1-\epsilon)\cdot D] \le \dfrac{4}{\epsilon^2 t}$
- Setting $t = 24/\epsilon^2$ gives us

$$\Pr[(1-\epsilon)\cdot D \le Y \le (1+\epsilon)\cdot D] \ge 1 - \frac{8}{\epsilon^2 t} = 2/3$$

Practice problem: how can we make the success probability much higher?

[1] replacing $1-\epsilon$ by $1+\epsilon$ and using Chebyshev

Space requirements and running time

$\boxed{\log m = b}$

- Total space used: $O\left(\frac{1}{\epsilon^2}\log m\right)$ bits

hash function

$O(\log m)$

stored $t = 24/\epsilon^2$

hash values

$O\left(\frac{1}{\epsilon^2}\cdot\log m\right)$

Space requirements and running time

- Total space used: $O\left(\frac{1}{\epsilon^2}\log m\right)$ bits
  - we stored $O(1/\epsilon^2)$ hash values each of $\log(m)$ bits
  - hash function only requires $O(\log m)$ bits to store.

Space requirements and running time

- Total space used: $O\left(\frac{1}{\epsilon^2}\log m\right)$ bits
  - we stored $O(1/\epsilon^2)$ hash values each of $\log(m)$ bits
  - hash function only requires $O(\log m)$ bits to store.
- Running time per operation: $O(\log(m) + 1/\epsilon^2)$ steps

Space requirements and running time

- Total space used: $O\left(\frac{1}{\epsilon^2}\log m\right)$ bits
  - we stored $O(1/\epsilon^2)$ hash values each of $\log(m)$ bits
  - hash function only requires $O(\log m)$ bits to store.
- Running time per operation: $O(\log(m) + 1/\epsilon^2)$ steps
  - compute hash in $O(\log m)$ time
  - Since we keep track of $O(1/\epsilon^2)$ elements, and need to update the list, this takes $O(1/\epsilon^2)$ time (though there are smarter ways)

Heavy hitters with weights

Example (Weighted heavy hitters)

- **Input stream:** $(a_1, w_1), \ldots, (a_N, w_N)$ tuples of integers from $\Sigma = [-2^b + 1, 2^b - 1]$, parameter $q \in \mathbb{N}$

Heavy hitters with weights

Example (Weighted heavy hitters)

- **Input stream:** $(a_1, w_1), \ldots, (a_N, w_N)$ tuples of integers from $\Sigma = [-2^b + 1, 2^b - 1]$, parameter $q \in \mathbb{N}$
  - Total weight

$$Q = \sum_{t=1}^{N} w_t$$

Heavy hitters with weights

Example (Weighted heavy hitters)

- **Input stream:** $(a_1, w_1), \ldots, (a_N, w_N)$ tuples of integers from $\Sigma = [-2^b + 1, 2^b - 1]$, parameter $q \in \mathbb{N}$
  - Total weight
$$Q = \sum_{t=1}^{N} w_t$$
  - Total weight of $e \in \Sigma$:
$$Q(e) = \sum_{t: a_t = e} w_t$$

Heavy hitters with weights

Example (Weighted heavy hitters)

- **Input stream:** $(a_1, w_1), \ldots, (a_N, w_N)$ tuples of integers from $\Sigma = [-2^b + 1, 2^b - 1]$, parameter $q \in \mathbb{N}$
  - Total weight
$$Q = \sum_{t=1}^{N} w_t$$
  - Total weight of $e \in \Sigma$:
$$Q(e) = \sum_{t:a_t=e} w_t$$
- **Task:** find all elements e such that $Q(e) \geq q$

Heavy hitters with weights

Example (Weighted heavy hitters)

- **Input stream:** $(a_1, w_1), \dots, (a_N, w_N)$ tuples of integers from $\Sigma = [-2^b + 1, 2^b - 1]$, parameter $q \in \mathbb{N}$
  - Total weight
$$Q = \sum_{t=1}^{N} w_t$$
  - Total weight of $e \in \Sigma$:
$$Q(e) = \sum_{t:a_t=e} w_t$$
- **Task:** find all elements e such that $Q(e) \geq q$
- **Constraint:** allowed to also output *false positives* (low hitters), but not allowed to miss any heavy hitter!

Weighted heavy hitters - algorithm setup

We will see an algorithm that gives us the following guarantees:

Weighted heavy hitters - algorithm setup

We will see an algorithm that gives us the following guarantees:

1. All heavy hitters are reported

Weighted heavy hitters - algorithm setup

We will see an algorithm that gives us the following guarantees:

1. All heavy hitters are reported
2. if $Q(e) \leq q - \epsilon \cdot Q$, then e is reported with probability at most δ
    - That is, have low probability of reporting a really low hitter

Weighted heavy hitters - algorithm setup

We will see an algorithm that gives us the following guarantees:

1. All heavy hitters are reported
2. if $Q(e) \leq q - \epsilon \cdot Q$, then e is reported with probability at most δ
   - That is, have low probability of reporting a really low hitter

Use 2-universal hash functions!

Weighted heavy hitters - algorithm setup

We will see an algorithm that gives us the following guarantees:

1. All heavy hitters are reported
2. if $Q(e) \leq q - \epsilon \cdot Q$, then e is reported with probability at most δ
    - That is, have low probability of reporting a really low hitter

Use 2-universal hash functions!

- k, ℓ are parameters to be chosen later

Weighted heavy hitters - algorithm setup

We will see an algorithm that gives us the following guarantees:

1. All heavy hitters are reported
2. if $Q(e) \leq q - \epsilon \cdot Q$, then e is reported with probability at most δ
    - That is, have low probability of reporting a really low hitter

Use 2-universal hash functions!

- k, ℓ are parameters to be chosen later
- Pick k hash functions $h_1, \ldots, h_k$ where $h_i : \Sigma \to [0, \ell - 1]$

Weighted heavy hitters - algorithm setup

We will see an algorithm that gives us the following guarantees:

1. All heavy hitters are reported
2. if $Q(e) \leq q - \epsilon \cdot Q$, then e is reported with probability at most δ
   - That is, have low probability of reporting a really low hitter

Use 2-universal hash functions!

- k, ℓ are parameters to be chosen later
- Pick k hash functions $h_1, \ldots, h_k$ where $h_i : \Sigma \to [0, \ell - 1]$
- Let's maintain $k \cdot \ell$ counters $C_{i,j}$, where each $C_{i,j}$ adds the weight of items that are mapped to j^{th} entry by the i^{th} hash function. Start with $C_{i,j} = 0$ for all $1 \leq i \leq k$ and $1 \leq j \leq \ell$.

Weighted heavy hitters - algorithm

- Given (a_t, w_t), for each $1 \leq i \leq k$ set $C_{i,h_i(a_t)} \leftarrow C_{i,h_i(a_t)} + w_t$.
- At the end,[2] report all elements e with

$$\min_{1 \leq i \leq k} C_{i,h_i(e)} \geq q$$

- Data structure as a table:

[2] In this version need to do second pass over data. But this can be fixed. Practice problem: fix this so that we can report on the fly.

Weighted heavy hitters - analysis

- Heavy hitter always reported, as all their counters are large

Weighted heavy hitters - analysis

- Heavy hitter always reported, as all their counters are large
- Need to show now that if e is not a heavy hitter, with high probability we will have one counter $C_{i,h_i(e)} < q$.

Weighted heavy hitters - analysis

- Heavy hitter always reported, as all their counters are large
- Need to show now that if e is not a heavy hitter, with high probability we will have one counter $C_{i,h_i(e)} < q$.
- If $Q(e) \leq q - \epsilon \cdot Q$, what is prob. e will be reported as heavy hitter?

Weighted heavy hitters - analysis

- Heavy hitter always reported, as all their counters are large
- Need to show now that if e is not a heavy hitter, with high probability we will have one counter $C_{i,h_i(e)} < q$.
- If $Q(e) \leq q - \epsilon \cdot Q$, what is prob. e will be reported as heavy hitter?
  - Look at counter $C_{i,h_i(e)}$. Since e is reported, must have $C_{i,h_i(e)} \geq q$

Weighted heavy hitters - analysis

- Heavy hitter always reported, as all their counters are large
- Need to show now that if e is not a heavy hitter, with high probability we will have one counter $C_{i,h_i(e)} < q$.
- If $Q(e) \leq q - \epsilon \cdot Q$, what is prob. e will be reported as heavy hitter?
  - Look at counter $C_{i,h_i(e)}$. Since e is reported, must have $C_{i,h_i(e)} \geq q$
  - Contribution from e is $Q(e) \leq q - \epsilon \cdot Q$. So other elements that map to $h_i(e)$ must have contributed $\geq \epsilon \cdot Q$.

Weighted heavy hitters - analysis

- Heavy hitter always reported, as all their counters are large
- Need to show now that if e is not a heavy hitter, with high probability we will have one counter $C_{i,h_i(e)} < q$.
- If $Q(e) \leq q - \epsilon \cdot Q$, what is prob. e will be reported as heavy hitter?
  - Look at counter $C_{i,h_i(e)}$. Since e is reported, must have $C_{i,h_i(e)} \geq q$
  - Contribution from e is $Q(e) \leq q - \epsilon \cdot Q$. So other elements that map to $h_i(e)$ must have contributed $\geq \epsilon \cdot Q$.
  - Let Z_i be the value of $C_{i,h_i(e)}$ that was added by other elements

Weighted heavy hitters - analysis

- Heavy hitter always reported, as all their counters are large
- Need to show now that if e is not a heavy hitter, with high probability we will have one counter $C_{i,h_i(e)} < q$.
- If $Q(e) \leq q - \epsilon \cdot Q$, what is prob. e will be reported as heavy hitter?
  - Look at counter $C_{i,h_i(e)}$. Since e is reported, must have $C_{i,h_i(e)} \geq q$
  - Contribution from e is $Q(e) \leq q - \epsilon \cdot Q$. So other elements that map to $h_i(e)$ must have contributed $\geq \epsilon \cdot Q$.
  - Let Z_i be the value of $C_{i,h_i(e)}$ that was added by other elements
  - h_i chosen from 2-universal hash family then probability that another element f is mapped to $h_i(e)$ is $\leq 1/\ell$.

Weighted heavy hitters - analysis

- Heavy hitter always reported, as all their counters are large
- Need to show now that if e is not a heavy hitter, with high probability we will have one counter $C_{i,h_i(e)} < q$.
- If $Q(e) \leq q - \epsilon \cdot Q$, what is prob. e will be reported as heavy hitter?
  - Look at counter $C_{i,h_i(e)}$. Since e is reported, must have $C_{i,h_i(e)} \geq q$
  - Contribution from e is $Q(e) \leq q - \epsilon \cdot Q$. So other elements that map to $h_i(e)$ must have contributed $\geq \epsilon \cdot Q$.
  - Let Z_i be the value of $C_{i,h_i(e)}$ that was added by other elements
  - h_i chosen from 2-universal hash family then probability that another element f is mapped to $h_i(e)$ is $\leq 1/\ell$.
  - Thus $\mathbb{E}[Z_i] \leq Q/\ell$. By Markov:

$$\Pr[Z_i \geq \epsilon \cdot Q] \leq \frac{\mathbb{E}[Z]}{\epsilon \cdot Q} \leq \frac{1}{\epsilon \ell}$$

Weighted heavy hitters - analysis

- Heavy hitter always reported, as all their counters are large
- Need to show now that if e is not a heavy hitter, with high probability we will have one counter $C_{i,h_i(e)} < q$.
- If $Q(e) \leq q - \epsilon \cdot Q$, what is prob. e will be reported as heavy hitter?
  - Look at counter $C_{i,h_i(e)}$. Since e is reported, must have $C_{i,h_i(e)} \geq q$
  - Contribution from e is $Q(e) \leq q - \epsilon \cdot Q$. So other elements that map to $h_i(e)$ must have contributed $\geq \epsilon \cdot Q$.
  - Let Z_i be the value of $C_{i,h_i(e)}$ that was added by other elements
  - h_i chosen from 2-universal hash family then probability that another element f is mapped to $h_i(e)$ is $\leq 1/\ell$.
  - Thus $\mathbb{E}[Z_i] \leq Q/\ell$. By Markov:

$$\Pr[Z_i \geq \epsilon \cdot Q] \leq \frac{\mathbb{E}[Z]}{\epsilon \cdot Q} \leq \frac{1}{\epsilon \ell}$$

  - Hash functions h_i chosen independently $\Rightarrow$

$$\Pr\left[\min_{1 \leq i \leq k} Z_i \geq \epsilon \cdot Q\right] \leq \left(\frac{1}{\epsilon \ell}\right)^k$$

Weighted heavy hitters - analysis

We have

$$\Pr\left[\min_{1\leq i\leq k} Z_i \geq \epsilon \cdot Q\right] \leq \left(\frac{1}{\epsilon\ell}\right)^k$$

Weighted heavy hitters - analysis

We have

$$\Pr\left[\min_{1\le i\le k} Z_i \ge \epsilon \cdot Q\right] \le \left(\frac{1}{\epsilon\ell}\right)^k$$

- Setting $\ell = 2/\epsilon$ and $k = \log(\delta)$ we get that probability above $\le \delta$.

Weighted heavy hitters - analysis

We have

$$\Pr\left[\min_{1\leq i\leq k} Z_i \geq \epsilon \cdot Q\right] \leq \left(\frac{1}{\epsilon\ell}\right)^k$$

- Setting $\ell = 2/\epsilon$ and $k = \log(\delta)$ we get that probability above $\leq \delta$.
- Space requirement for counters $O(1/\epsilon \cdot \log(1/\delta))$

Weighted heavy hitters - analysis

We have

$$\Pr\left[\min_{1\le i\le k} Z_i \ge \epsilon \cdot Q\right] \le \left(\frac{1}{\epsilon\ell}\right)^k$$

- Setting $\ell = 2/\epsilon$ and $k = \log(\delta)$ we get that probability above $\le \delta$.
- Space requirement for counters $O(1/\epsilon \cdot \log(1/\delta))$
- Space required to store all hash functions and evaluation time $O(k \cdot \ell)$

Acknowledgement

- Lecture based largely on Lap Chi's notes and David Woodruff's notes.
- See Lap Chi's notes at
  https://cs.uwaterloo.ca/~lapchi/cs466/notes/L05.pdf
- See David's notes at
  https://www.cs.cmu.edu/~15451-s20/lectures/lec6.pdf