Karlsruhe Institute of Technology

Probability and Computing – Streaming

Stefan Walzer | WS 2025/2026



Content

What is a Streaming Algorithm?

- looong input data stream $(a_1, \ldots, a_m) \in [n]^m$ can only be read *once* from left to right
- goal: approximate some value $F = F(a_1, \ldots, a_m)$ with small relative error ε and failure probability δ.
  $\hookrightarrow$ streaming algorithms are approximation algorithms
- challenge: use less *space* than exact algorithm (in particular: cannot store $(a_1, \ldots, a_m)$).
  $\hookrightarrow$ don't care about running time

What is a Streaming Algorithm?

- looong input data stream $(a_1, \dots, a_m) \in [n]^m$ can only be read *once* from left to right
- goal: approximate some value $F = F(a_1, \dots, a_m)$ with small relative error ε and failure probability δ.
  $\hookrightarrow$ streaming algorithms are approximation algorithms
- challenge: use less *space* than exact algorithm (in particular: cannot store $(a_1, \dots, a_m)$).
  $\hookrightarrow$ don't care about running time

Formally, a streaming algorithm is given by three algorithms init, update and result used as follows:

$Z \leftarrow \text{init}()$
for $i = 1$ **to** m **do**
    $Z \leftarrow \text{update}(Z, a_i)$
return $\text{result}(Z)$

Its space complexity is the space required for Z.

What is a Streaming Algorithm?

- looong input data stream $(a_1, \dots, a_m) \in [n]^m$ can only be read *once* from left to right
- goal: approximate some value $F = F(a_1, \dots, a_m)$ with small relative error ε and failure probability δ.
  $\hookrightarrow$ streaming algorithms are approximation algorithms
- challenge: use less *space* than exact algorithm (in particular: cannot store $(a_1, \dots, a_m)$).
  $\hookrightarrow$ don't care about running time

Formally, a streaming algorithm is given by three algorithms init, update and result used as follows:

$Z \leftarrow$ init()
for $i = 1$ **to** m **do**
  $Z \leftarrow$ update(Z, a_i)
return result(Z)

Its space complexity is the space required for Z.

Today's Motivating Examples

A. Router approximately counts traffic over each connection.
  $\hookrightarrow$ maybe: detect anomalies related to DDoS

B. Website approximately counts number of unique users visiting a resource.

What is a Streaming Algorithm?

- looong input data stream $(a_1, \ldots, a_m) \in [n]^m$ can only be read *once* from left to right
- goal: approximate some value $F = F(a_1, \ldots, a_m)$ with small relative error ε and failure probability δ.
  $\hookrightarrow$ streaming algorithms are approximation algorithms
- challenge: use less *space* than exact algorithm (in particular: cannot store $(a_1, \ldots, a_m)$).
  $\hookrightarrow$ don't care about running time

Formally, a streaming algorithm is given by three algorithms init, update and result used as follows:

$Z \leftarrow$ init()
for $i = 1$ **to** m **do**
  $Z \leftarrow$ update(Z, a_i)
return result(Z)

Its space complexity is the space required for Z.

Today's Motivating Examples

A Router approximately counts traffic over each connection.
$\hookrightarrow$ maybe: detect anomalies related to DDoS

B Website approximately counts number of unique users visiting a resource.

Today's Formal Results

A Approximate $F_1(a_1, \ldots, a_m) = m$ in expected space $\frac{1}{\varepsilon^2 \delta} \log \log m$.

B Approximate $F_0(a_1, \ldots, a_m) = |\{a_1, \ldots, a_m\}|$ in expected space $\frac{1}{\varepsilon^2} \log(n) \cdot \log(m/\delta)$.

Content

Attempt I: Naive Counting

Naive Counting

Algorithm init**:**
- $Z \leftarrow 0$
- **return** Z

Algorithm update(Z, a)**:**
- $Z \leftarrow Z + 1$
- **return** Z

Algorithm result(Z)**:**
- **return** Z

Attempt I: Naive Counting

Naive Counting

Algorithm init**:**
- $Z \leftarrow 0$
- **return** Z

Algorithm update(Z, a)**:**
- $Z \leftarrow Z + 1$
- **return** Z

Algorithm result(Z)**:**
- **return** Z

Observations on Naive counting

- No errors ($\varepsilon = \delta = 0$).
- Requires $\lceil \log(m+1) \rceil$ bits of memory.
- No *deterministic* algorithm can use less space

Attempt I: Naive Counting

Karlsruhe Institute of Technology

Naive Counting

Algorithm init**:**
  $Z \leftarrow 0$
  return Z

Algorithm update(Z, a)**:**
  $Z \leftarrow Z + 1$
  return Z

Algorithm result(Z)**:**
  return Z

Observations on Naive counting

- No errors ($\varepsilon = \delta = 0$).
- Requires $\lceil \log(m + 1) \rceil$ bits of memory.
- No *deterministic* algorithm can use less space
  - Would have to "reuse" a state Z.
  - Is then trapped in an infinite loop.
  - Result arbitrarily far off if m large enough.

Attempt II: Lossy Counting

Lossy Counting, parameter p

Algorithm init**:**
  $Z \leftarrow 0$
  return Z

Algorithm update(Z, a)**:**
  with probability p **do**
    $Z \leftarrow Z + 1$
  return Z

Algorithm result(Z)**:**
  return Z/p

Attempt II: Lossy Counting

Approximate Counting

- stream $(a_1, \dots, a_m)$
- want $F_1 = m$

Lossy Counting, parameter p

Algorithm init**:**
    $Z \leftarrow 0$
    return Z

Algorithm update(Z, a)**:**
    with probability p **do**
        $Z \leftarrow Z + 1$
    return Z

Algorithm result(Z)**:**
    return Z/p

Analysis (Exercise)

For any $p \in (0, 1]$ we have

- $\mathbb{E}[\text{result}] = m$
- $\Pr[|\text{result} - m| \le \varepsilon m] \ge 1 - 2\exp(-\varepsilon^2 pm/3)$.
- $\mathbb{E}[\text{space}] \le \log_2(1 + mp) + 1$.

Attempt II: Lossy Counting

Lossy Counting, parameter p

Algorithm init**:**
- $Z \leftarrow 0$
- **return** Z

Algorithm update(Z, a)**:**
- **with probability** p **do**
  - $Z \leftarrow Z + 1$
- **return** Z

Algorithm result(Z)**:**
- **return** Z/p

Analysis (Exercise)

For any $p \in (0, 1]$ we have

- $\mathbb{E}[\text{result}] = m$
- $\Pr[|\text{result} - m| \leq \varepsilon m] \geq 1 - 2\exp(-\varepsilon^2 pm/3)$.
- $\mathbb{E}[\text{space}] \leq \log_2(1 + mp) + 1$.

Corollary

By choosing $p = \frac{3}{\varepsilon^2 m} \ln(2/\delta)$ we get

$$\Pr[\text{fail}] \leq \delta \text{ and } \mathbb{E}[\text{space}] \leq \mathcal{O}(\log(\tfrac{1}{\varepsilon}) + \log\log(1/\delta))).$$

Karlsruhe Institute of Technology

Attempt II: Lossy Counting

Lossy Counting, parameter p

Algorithm init**:**
- $Z \leftarrow 0$
- **return** Z

Algorithm update(Z, a)**:**
- **with probability** p **do**
  - $Z \leftarrow Z + 1$
- **return** Z

Algorithm result(Z)**:**
- **return** Z/p

Analysis (Exercise)

For any $p \in (0, 1]$ we have

- $\mathbb{E}[\text{result}] = m$
- $\Pr[|\text{result} - m| \leq \varepsilon m] \geq 1 - 2\exp(-\varepsilon^2 pm/3)$.
- $\mathbb{E}[\text{space}] \leq \log_2(1 + mp) + 1$.

Corollary

By choosing $p = \frac{3}{\varepsilon^2 m} \ln(2/\delta)$ we get

$$\Pr[\text{fail}] \leq \delta \text{ and } \mathbb{E}[\text{space}] \leq \mathcal{O}(\log(\tfrac{1}{\varepsilon}) + \log\log(1/\delta)).$$

Serious Objection

Correctly choosing p requires already knowing m.

(or at least the order of magnitude of m)

Attempt III: Morris' Algorithm

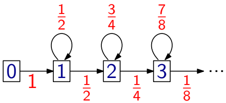
Approximate Counting
- stream $(a_1, \ldots, a_m)$
- want $F_1 = m$

Morris' Algorithm

Algorithm init**:**
- $Z \leftarrow 0$
- **return** Z

Algorithm update(Z, a)**:**
- **with probability** 2^{-Z} **do**
  - $Z \leftarrow Z + 1$
- **return** Z

Algorithm result(Z)**:**
- **return** $2^Z - 1$



states of Z
transition probabilities

Attempt III: Morris' Algorithm

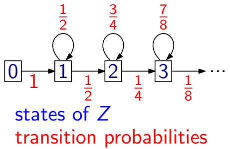
Approximate Counting
- stream $(a_1, \dots, a_m)$
- want $F_1 = m$

Morris' Algorithm

Algorithm init**:**
- $Z \leftarrow 0$
- **return** Z

Algorithm update(Z, a)**:**
- **with probability** 2^{-Z} **do**
  - $Z \leftarrow Z + 1$
- **return** Z

Algorithm result(Z)**:**
- **return** $2^Z - 1$



Lemma: Morris' Algorithm is an *Unbiased Estimator*

$\mathbb{E}[\text{result}] = m$.

Proof

Let Z_i for $i \in [m]$ denote the value of Z after i updates.

Attempt III: Morris' Algorithm

Approximate Counting

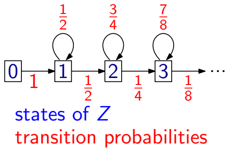
- stream $(a_1, \dots, a_m)$
- want $F_1 = m$

Morris' Algorithm

Algorithm init**:**
- $Z \leftarrow 0$
- **return** Z

Algorithm update(Z, a)**:**
- **with probability** 2^{-Z} **do**
  - $Z \leftarrow Z + 1$
- **return** Z

Algorithm result(Z)**:**
- **return** $2^Z - 1$



Lemma: Morris' Algorithm is an *Unbiased Estimator*

$\mathbb{E}[\text{result}] = m$.

Proof

Let Z_i for $i \in [m]$ denote the value of Z after i updates.
Consider the expected change to 2^Z in one step...

- ... conditioned on a current value $j \in \mathbb{N}$:

$$\mathbb{E}[2^{Z_{i+1}} - 2^{Z_i} \mid Z_i = j] = 2^{-j} \cdot (2^{j+1} - 2^j) + (1 - 2^{-j}) \cdot \underbrace{(2^j - 2^j)}_{=0} = 2 - 1 = 1.$$

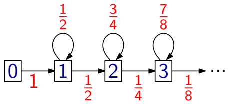
Attempt III: Morris' Algorithm

Morris' Algorithm

Algorithm init:
- $Z \leftarrow 0$
- **return** Z

Algorithm update(Z, a):
- **with probability** 2^{-Z} **do**
  - $Z \leftarrow Z + 1$
- **return** Z

Algorithm result(Z):
- **return** $2^Z - 1$



states of Z
transition probabilities

Lemma: Morris' Algorithm is an *Unbiased Estimator*

$\mathbb{E}[\text{result}] = m$.

Proof

Let Z_i for $i \in [m]$ denote the value of Z after i updates.
Consider the expected change to 2^Z in one step...

- ... conditioned on a current value $j \in \mathbb{N}$:

$$\mathbb{E}[2^{Z_{i+1}} - 2^{Z_i} \mid Z_i = j] = 2^{-j} \cdot (2^{j+1} - 2^j) + (1 - 2^{-j}) \cdot \underbrace{(2^j - 2^j)}_{=0} = 2 - 1 = 1.$$

- ... unconditionally:

$$\mathbb{E}[2^{Z_{i+1}} - 2^{Z_i}] \overset{\text{LTE}}{=} \sum_{j \geq 0} \Pr[Z_i = j] \cdot \underbrace{\mathbb{E}[2^{Z_{i+1}} - 2^{Z_i} \mid Z_i = j]}_{=1} = \sum_{j \geq 0} \Pr[Z_i = j] = 1.$$

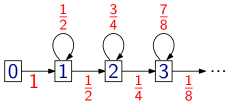
Attempt III: Morris' Algorithm

Morris' Algorithm

Algorithm init:
- $Z \leftarrow 0$
- **return** Z

Algorithm update(Z, a):
- **with probability** 2^{-Z} **do**
  - $Z \leftarrow Z + 1$
- **return** Z

Algorithm result(Z):
- **return** $2^Z - 1$



states of Z
transition probabilities

Lemma: Morris' Algorithm is an *Unbiased Estimator*

$\mathbb{E}[\text{result}] = m$.

Proof

Let Z_i for $i \in [m]$ denote the value of Z after i updates.
Consider the expected change to 2^Z in one step...

- ... conditioned on a current value $j \in \mathbb{N}$:

$$\mathbb{E}[2^{Z_{i+1}} - 2^{Z_i} \mid Z_i = j] = 2^{-j} \cdot (2^{j+1} - 2^j) + (1 - 2^{-j}) \cdot \underbrace{(2^j - 2^j)}_{=0} = 2 - 1 = 1.$$

- ... unconditionally:

$$\mathbb{E}[2^{Z_{i+1}} - 2^{Z_i}] \overset{\text{LTE}}{=} \sum_{j \geq 0} \Pr[Z_i = j] \cdot \underbrace{\mathbb{E}[2^{Z_{i+1}} - 2^{Z_i} \mid Z_i = j]}_{=1} = \sum_{j \geq 0} \Pr[Z_i = j] = 1.$$

Hence:

$$\mathbb{E}[\text{result}] = \mathbb{E}[2^{Z_m} - 1] = \mathbb{E}[2^{Z_m} - 2^{Z_0}] = \mathbb{E}[\sum_{i=1}^{m} 2^{Z_{i+1}} - 2^{Z_i}] = \sum_{i=1}^{m} \underbrace{\mathbb{E}[2^{Z_{i+1}} - 2^{Z_i}]}_{=1} = m.$$

Attempt III: Morris' Algorithm

Approximate Counting

- stream $(a_1, \ldots, a_m)$
- want $F_1 = m$

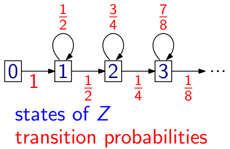
Karlsruhe Institute of Technology

Morris' Algorithm

Algorithm init**:**
- $Z \leftarrow 0$
- **return** Z

Algorithm update(Z, a)**:**
- **with probability** 2^{-Z} **do**
  - $Z \leftarrow Z + 1$
- **return** Z

Algorithm result(Z)**:**
- **return** $2^Z - 1$



Lemma 1: Worryingly large Variance

$\mathrm{Var}(2^{Z_i}) = \frac{i^2 - i}{2} = \Theta(i^2)$.

Lemma 2

$\mathbb{E}[2^{2Z_i}] = \frac{3i(i+1)}{2} + 1$.

Proof of Lemma 1 using Lemma 2.

$$\mathrm{Var}(2^{Z_i}) = \mathbb{E}[2^{2Z_i}] - \mathbb{E}[2^{Z_i}]^2 \overset{\text{Lem. 2}}{=} \frac{3i(i+1)}{2} + 1 - (i+1)^2 = \frac{3}{2}i^2 - i^2 \pm \mathcal{O}(i) = \Theta(i^2)$$

Attempt III: Morris' Algorithm

Approximate Counting

- stream $(a_1, \dots, a_m)$
- want $F_1 = m$

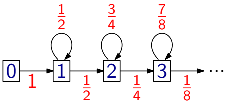
Karlsruhe Institute of Technology

Morris' Algorithm

Algorithm init**:**
- $Z \leftarrow 0$
- **return** Z

Algorithm update(Z, a)**:**
- **with probability** 2^{-Z} **do**
  - $Z \leftarrow Z + 1$
- **return** Z

Algorithm result(Z)**:**
- **return** $2^Z - 1$



states of Z
transition probabilities

Lemma 1: Worryingly large Variance

$\mathrm{Var}(2^{Z_i}) = \frac{i^2 - i}{2} = \Theta(i^2)$.

Lemma 2

$\mathbb{E}[2^{2Z_i}] = \frac{3i(i+1)}{2} + 1$.

Proof of Lemma 2.

For $i \in \{0, 1\}$ ✓. Let now $i \geq 1$. Note $\Pr[Z_{i+1} = 0] = \Pr[Z_i = 0] = 0$.

$$\mathbb{E}[2^{2Z_{i+1}}] = \sum_{j \geq 1} 2^{2j} \Pr[Z_{i+1} = j] = \sum_{j \geq 1} 2^{2j} (\Pr[Z_i = j-1] \cdot 2^{-j+1} + \Pr[Z_i = j] \cdot (1 - 2^{-j}))$$

$$= \sum_{j \geq 1} 2^{j+1} \Pr[Z_i = j-1] + \sum_{j \geq 1} 2^{2j} \Pr[Z_i = j] - \sum_{j \geq 1} 2^j \Pr[Z_i = j]$$

$$= 4 \sum_{j \geq 0} 2^j \Pr[Z_i = j] + \sum_{j \geq 0} 2^{2j} \Pr[Z_i = j] - \sum_{j \geq 0} 2^j \Pr[Z_i = j]$$

$$= 4\mathbb{E}[2^{Z_i}] + \mathbb{E}[2^{2Z_i}] - \mathbb{E}[2^{Z_i}] = 3\mathbb{E}[2^{Z_i}] + \mathbb{E}[2^{2Z_i}] = 3(i+1) + \mathbb{E}[2^{2Z_i}]$$

$$\overset{\text{Ind.}}{=} 3(i+1) + \frac{3i(i+1)}{2} + 1 = \frac{3(i+2)(i+1)}{2} + 1. \quad \square$$

Space

Expected Space

$$\mathbb{E}[\text{space}] \le \mathbb{E}[\lceil \log_2(1+Z_m) \rceil] \le 1 + \mathbb{E}[\log_2(1+Z_m)] = 1 + \mathbb{E}[\log_2(1+\log_2(2^{Z_m}))]$$
$$\overset{(\star)}{\le} 1 + \log_2(1+\log_2(\mathbb{E}[2^{Z_m}])) = 1 + \log_2(1+\log_2(m+1)) = \Theta(\log\log m).$$

$(\star)$ uses Jensen's inequality that you'll prove as an exercise.

Space

Expected Space

$$\mathbb{E}[\text{space}] \le \mathbb{E}[\lceil \log_2(1+Z_m) \rceil] \le 1 + \mathbb{E}[\log_2(1+Z_m)] = 1 + \mathbb{E}[\log_2(1+\log_2(2^{Z_m}))]$$

$$\overset{(\star)}{\le} 1 + \log_2(1+\log_2(\mathbb{E}[2^{Z_m}])) = 1 + \log_2(1+\log_2(m+1)) = \Theta(\log\log m).$$

$(\star)$ uses Jensen's inequality that you'll prove as an exercise.

Interim Conclusion: Morris is not good enough *yet*

- $\mathbb{E}[\text{result}] = m$ ✓ unbiased estimator
- $\mathbb{E}[\text{space}] = \mathcal{O}(\log\log m)$ ✓ highly space efficient
- $\text{Var}(\text{result}) = \Theta(m^2)$ ✗
  - Standarddeviation $\Theta(m)$
    $\rightsquigarrow$ typically right order of magnitude, but not better.

Morris$^+$: Use many copies of Morris' Algorithm

Karlsruhe Institute of Technology

Theorem

Consider a streaming algorithm that maintains a sequence $Z = (Z_1, \ldots, Z_s)$ of independent Morris-counters and returns $\text{result}(Z) := \frac{\text{result}(Z_1)+\cdots+\text{result}(Z_s)}{s}$. For $s = \frac{1}{\varepsilon^2\delta}$ we obtain

- $\mathbb{E}[\text{result}(Z)] = m$ and $\mathbb{E}[\text{space}] = \mathcal{O}(\frac{1}{\varepsilon^2\delta} \log\log m)$
- $\Pr[|\text{result}(Z) - m| \leq \varepsilon m] = 1 - \mathcal{O}(\delta)$.

Morris$^+$: Use many copies of Morris' Algorithm

Karlsruhe Institute of Technology

Theorem

Consider a streaming algorithm that maintains a sequence $Z = (Z_1, \ldots, Z_s)$ of independent Morris-counters and returns $\text{result}(Z) := \frac{\text{result}(Z_1)+\cdots+\text{result}(Z_s)}{s}$. For $s = \frac{1}{\varepsilon^2\delta}$ we obtain

- $\mathbb{E}[\text{result}(Z)] = m$ and $\mathbb{E}[\text{space}] = \mathcal{O}(\frac{1}{\varepsilon^2\delta} \log\log m)$
- $\Pr[|\text{result}(Z) - m| \leq \varepsilon m] = 1 - \mathcal{O}(\delta)$.

Reminder on Variance

If X, Y are independent random variables and $s > 0$ then

- $\text{Var}(sX) = s^2\,\text{Var}(X)$
- $\text{Var}(X + Y) = \text{Var}(X) + \text{Var}(Y)$

Proof of Concentration using Chebyshev

$$\text{Var}(\text{result}(Z)) = \text{Var}\Big(\frac{1}{s}\sum_{i=1}^{s}\text{result}(Z_i)\Big) = \frac{1}{s^2}\,\text{Var}\Big(\sum_{i=1}^{s}\text{result}(Z_i)\Big)$$

$$= \frac{1}{s^2}\sum_{i=1}^{s}\text{Var}(\text{result}(Z_i)) = \frac{s}{s^2}\,\text{Var}(\text{result}(Z_1)) = \frac{1}{s}\Theta(m^2) = \Theta(m^2/s).$$

Chebyschev:

$$\Pr[X - \mathbb{E}[X] > c] \leq \frac{\text{Var}(X)}{c^2}.$$

$$\Pr[\text{fail}] = \Pr[|\text{result}(Z) - m| > \varepsilon m] = \Pr[|\text{result}(Z) - \mathbb{E}[\text{result}(Z)]| > \varepsilon m] \leq \frac{\text{Var}(\text{result}(Z))}{\varepsilon^2 m^2} = \Theta(1/(\varepsilon^2 s)) = \Theta(\delta). \quad \square$$

Morris*: Use a different base in Morris' Algorithm

Karlsruhe Institute of Technology

Morris with base $1 < \rho \ll 2$

- In every update: increment Z with probability ρ^{-Z}.
- In the end: return $\frac{\rho^Z - 1}{\rho - 1}$.

Modified Analysis (There is a bug in here, I'll fix it till the next lecture)

Show similarly to before:

- $\mathbb{E}[\text{result}] = m$
- $\text{Var}(\text{result}) = \Theta(\frac{m^2}{\rho - 1})$

Choosing $\rho = 1 + \varepsilon^2\delta$ gives:

- $\Pr[|\text{result} - m| > \varepsilon m] = \mathcal{O}(\delta)$.
- $\mathbb{E}[\text{space}] = \mathcal{O}(\log \log m + \log \frac{1}{\delta\varepsilon})$.

Content

History

Counting Distinct Elements

- stream $(a_1, \ldots, a_m) \in [n]^m$
- want $F_0 = |\{a_1, \ldots, a_m\}|$

Karlsruhe Institute of Technology

Remark: CVM is not well-known

Popular line of algorithms for F_0 by Philippe Flajolet et al:

- ~~1984: Flajolet-Martin~~ (deprecated)
  ↪ https://en.wikipedia.org/wiki/Flajolet-Martin_algorithm
- ~~2003: LogLog~~ (deprecated)
- 2007: HyperLogLog
  ↪ `https://en.wikipedia.org/wiki/HyperLogLog`

The CVM-Algorithm

- 2022: European Symposium on *Simplicity* in Algorithms 2022
  ↪ „Distinct Elements in Streams: An Algorithm for the (Text) Book"
- is a bit worse than HyperLogLog
- is easier to analyse than HyperLogLog

Next: We develop CVM in three steps.

Attempt I: Naively storing the set

Naive Storing

Algorithm init**:**
- $Z \leftarrow \varnothing$
- **return** Z

Algorithm update(Z, a)**:**
- $Z \leftarrow Z \cup \{a\}$
- **return** Z

Algorithm result(Z)**:**
- **return** $|Z|$

Observation

Naively storing the set requires $\Omega(F_0 \cdot \log n)$ bits.

Attempt II: Storing the set lossily

LossyStore, parameter p

Algorithm init**:**
- $Z \leftarrow \varnothing$
- **return** Z

Algorithm update(Z, a)**:**
- $Z \leftarrow Z \setminus \{a\}$
- **with probability** p **do**
  - $Z \leftarrow Z \cup \{a\}$
- **return** Z

Algorithm result(Z)**:**
- **return** $|Z|$/p;

Attempt II: Storing the set lossily

LossyStore, parameter p

Algorithm init**:**
- $Z \leftarrow \varnothing$
- **return** Z

Algorithm update(Z, a)**:**
- $Z \leftarrow Z \setminus \{a\}$
- **with probability** p **do**
  - $Z \leftarrow Z \cup \{a\}$
- **return** Z

Algorithm result(Z)**:**
- **return** $|Z|$/p;

Analysis

Let $Z_0, \ldots, Z_m$ be the states of Z over time. Invariant: Each $a \in \{a_1, \ldots, a_i\}$ is in Z_i independently with probability p. Hence $|Z_m| \sim \mathrm{Bin}(F_0, p)$.

Attempt II: Storing the set lossily

LossyStore, parameter p

Algorithm init**:**
- $Z \leftarrow \varnothing$
- **return** Z

Algorithm update(Z, a)**:**
- $Z \leftarrow Z \setminus \{a\}$
- **with probability** p **do**
  - $Z \leftarrow Z \cup \{a\}$
- **return** Z

Algorithm result(Z)**:**
- **return** $|Z|$/p;

Analysis

Let $Z_0, \ldots, Z_m$ be the states of Z over time. Invariant: Each $a \in \{a_1, \ldots, a_i\}$ is in Z_i independently with probability p. Hence $|Z_m| \sim \mathrm{Bin}(F_0, p)$.

- $\mathbb{E}[\text{result}] = \mathbb{E}[|Z_m|/p] = \mathbb{E}[|Z_m|]/p = F_0 p/p = F_0$.
  $\hookrightarrow$ result is *unbiased estimator* of F_0.

Attempt II: Storing the set lossily

Counting Distinct Elements

- stream $(a_1, \ldots, a_m) \in [n]^m$
- want $F_0 = |\{a_1, \ldots, a_m\}|$

LossyStore, parameter p

Algorithm init**:**
  $Z \leftarrow \varnothing$
  return Z

Algorithm update(Z, a)**:**
  $Z \leftarrow Z \setminus \{a\}$
  with probability p **do**
    $Z \leftarrow Z \cup \{a\}$
  return Z

Algorithm result(Z)**:**
  return $|Z|$/p;

Chernoff for $X \sim \text{Bin}(n, p)$

$\Pr[|X - \mathbb{E}[X]| > \varepsilon\mathbb{E}[X]] \leq 2\exp(-\varepsilon^2\mathbb{E}[X]/3)$.

Analysis

Let $Z_0, \ldots, Z_m$ be the states of Z over time. Invariant: Each $a \in \{a_1, \ldots, a_i\}$ is in Z_i independently with probability p. Hence $|Z_m| \sim \text{Bin}(F_0, p)$.

- $\mathbb{E}[\text{result}] = \mathbb{E}[|Z_m|/p] = \mathbb{E}[|Z_m|]/p = F_0 p/p = F_0$.
  $\hookrightarrow$ result is *unbiased estimator* of F_0.
- $\Pr[\text{fail}] = \Pr[|\text{result} - F_0| > \varepsilon F_0] = \Pr[\big||Z_m|/p - F_0\big| > \varepsilon F_0]$

$$= \Pr[\big||Z_m| - pF_0\big| > \varepsilon pF_0] = \Pr[\big||Z_m| - \mathbb{E}[|Z_m|]\big| > \varepsilon\mathbb{E}[|Z_m|]]$$

$$\overset{\text{Chern.}}{\leq} 2\exp(-\varepsilon^2\mathbb{E}[|Z_m|]/3) = 2\exp(-\varepsilon^2 pF_0/3).$$

  $\hookrightarrow$ choose $p = p_\delta := \frac{3\log(2/\delta)}{\varepsilon^2 F_0}$ for $\Pr[\text{fail}] \leq \delta$.

Attempt II: Storing the set lossily

Counting Distinct Elements

- stream $(a_1, \ldots, a_m) \in [n]^m$
- want $F_0 = |\{a_1, \ldots, a_m\}|$

Karlsruhe Institute of Technology

LossyStore, parameter p

Algorithm init**:**
  $Z \leftarrow \varnothing$
  return Z

Algorithm update(Z, a)**:**
  $Z \leftarrow Z \setminus \{a\}$
  with probability p **do**
    $Z \leftarrow Z \cup \{a\}$
  return Z

Algorithm result(Z)**:**
  return $|Z|$/p;

Analysis

Let $Z_0, \ldots, Z_m$ be the states of Z over time. Invariant: Each $a \in \{a_1, \ldots, a_i\}$ is in Z_i independently with probability p. Hence $|Z_m| \sim \text{Bin}(F_0, p)$.

- $\mathbb{E}[\text{result}] = \mathbb{E}[|Z_m|/p] = \mathbb{E}[|Z_m|]/p = F_0 p/p = F_0$.
  $\hookrightarrow$ result is *unbiased estimator* of F_0.
- $\Pr[\text{fail}] = \Pr[|\text{result} - F_0| > \varepsilon F_0] = \Pr\big[\big||Z_m|/p - F_0\big| > \varepsilon F_0\big]$
  $= \Pr\big[\big||Z_m| - pF_0\big| > \varepsilon p F_0\big] = \Pr\big[\big||Z_m| - \mathbb{E}[|Z_m|]\big| > \varepsilon \mathbb{E}[|Z_m|]\big]$
  $\overset{\text{Chern.}}{\leq} 2\exp(-\varepsilon^2 \mathbb{E}[|Z_m|]/3) = 2\exp(-\varepsilon^2 p F_0/3)$.
  $\hookrightarrow$ choose $p = p_\delta := \frac{3\log(2/\delta)}{\varepsilon^2 F_0}$ for $\Pr[\text{fail}] \leq \delta$.
- Expected space *in the end* for $p = p_\delta$ (⚠ $\neq$ peak space consumption)
  $\mathbb{E}[|Z_m| \cdot \mathcal{O}(\log n)] = F_0 p_\delta \cdot \mathcal{O}(\log n) = \mathcal{O}(\frac{\log(1/\delta)}{\varepsilon^2} \log n)$.

Chernoff for $X \sim \text{Bin}(n, p)$

$\Pr[|X - \mathbb{E}[X]| > \varepsilon \mathbb{E}[X]] \leq 2\exp(-\varepsilon^2 \mathbb{E}[X]/3)$.

Attempt II: Storing the set lossily

Counting Distinct Elements

- stream $(a_1, \ldots, a_m) \in [n]^m$
- want $F_0 = |\{a_1, \ldots, a_m\}|$

Karlsruhe Institute of Technology

LossyStore, parameter p

Algorithm init**:**
- $Z \leftarrow \varnothing$
- **return** Z

Algorithm update(Z, a)**:**
- $Z \leftarrow Z \setminus \{a\}$
- **with probability** p **do**
  - $Z \leftarrow Z \cup \{a\}$
- **return** Z

Algorithm result(Z)**:**
- **return** $|Z|$/p;

Chernoff for $X \sim \mathrm{Bin}(n, p)$

$\Pr[|X - \mathbb{E}[X]| > \varepsilon\mathbb{E}[X]] \leq 2\exp(-\varepsilon^2\mathbb{E}[X]/3)$.

Analysis

Let $Z_0, \ldots, Z_m$ be the states of Z over time. Invariant: Each $a \in \{a_1, \ldots, a_i\}$ is in Z_i independently with probability p. Hence $|Z_m| \sim \mathrm{Bin}(F_0, p)$.

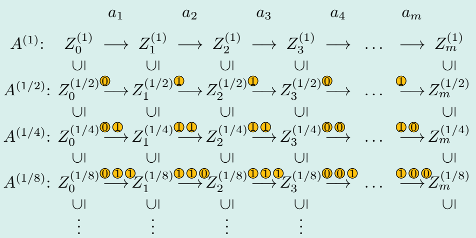
- $\mathbb{E}[\text{result}] = \mathbb{E}[|Z_m|/p] = \mathbb{E}[|Z_m|]/p = F_0 p/p = F_0$.
  $\hookrightarrow$ result is *unbiased estimator* of F_0.
- $\Pr[\text{fail}] = \Pr[|\text{result} - F_0| > \varepsilon F_0] = \Pr[\big||Z_m|/p - F_0\big| > \varepsilon F_0]$
  $= \Pr[\big||Z_m| - pF_0\big| > \varepsilon pF_0] = \Pr[\big||Z_m| - \mathbb{E}[|Z_m|]\big| > \varepsilon\mathbb{E}[|Z_m|]]$
  $\overset{\text{Chern.}}{\leq} 2\exp(-\varepsilon^2\mathbb{E}[|Z_m|]/3) = 2\exp(-\varepsilon^2 pF_0/3)$.
  $\hookrightarrow$ choose $p = p_\delta := \frac{3\log(2/\delta)}{\varepsilon^2 F_0}$ for $\Pr[\text{fail}] \leq \delta$.
- Expected space *in the end* for $p = p_\delta$ (⚠ $\neq$ peak space consumption)
  $\mathbb{E}[|Z_m| \cdot \mathcal{O}(\log n)] = F_0 p_\delta \cdot \mathcal{O}(\log n) = \mathcal{O}(\frac{\log(1/\delta)}{\varepsilon^2}\log n)$.

Serious Objection: Need to know F_0 to choose p

- for $p \gg p_\delta$: space is wasted
- for $p \ll p_\delta$: failure becomes likely

Attempt III: Adjust lossiness dynamically

Consider $A^{(p)} := \text{LossyStore}(p)$ with states $Z_0^{(p)}, \ldots, Z_m^{(p)}$ for $p \in \{1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \ldots\}$.



Coupling between executions of $A^{(p)}$

- $A^{(p/2)}$ uses coin tosses of $A^{(p)}$ and one more.
  "$A^{(p/2)}$ keeps half of what $A^{(p)}$ keeps."

Attempt III: Adjust lossiness dynamically

Counting Distinct Elements

- stream $(a_1, \dots, a_m) \in [n]^m$
- want $F_0 = |\{a_1, \dots, a_m\}|$

Karlsruhe Institute of Technology

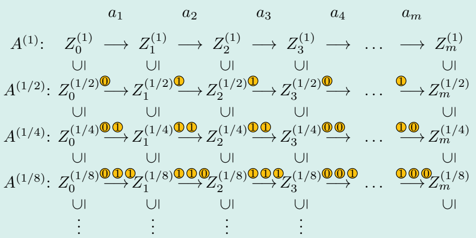
CVM, parameter T

Algorithm init:
- $Z \leftarrow \varnothing$
- $P \leftarrow 1$
- **return** (P, Z)

Algorithm update$((P, Z), a)$:
- $Z \leftarrow Z \setminus \{a\}$
- **with probability** P **do**
  - $Z \leftarrow Z \cup \{a\}$
- **while** $|Z| \geq T$ **do** // shrink
  - $Z' \leftarrow \varnothing$
  - **for** $a \in Z$ **do**
    - **with probability** $1/2$ **do**
      - $Z' \leftarrow Z' \cup \{a\}$
  - $(Z, P) \leftarrow (Z', P/2)$
- **return** (P, Z)

Algorithm result$((P, Z))$:
- **return** $|Z|/P$

Consider $A^{(p)} := \text{LossyStore}(p)$ with states $Z_0^{(p)}, \dots, Z_m^{(p)}$ for $p \in \{1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \dots\}$.



Coupling between executions of $A^{(p)}$

- $A^{(p/2)}$ uses coin tosses of $A^{(p)}$ and one more.
  "$A^{(p/2)}$ keeps half of what $A^{(p)}$ keeps."

Attempt III: Adjust lossiness dynamically

CVM, parameter T

Algorithm init:
- $Z \leftarrow \varnothing$
- $P \leftarrow 1$
- **return** (P, Z)

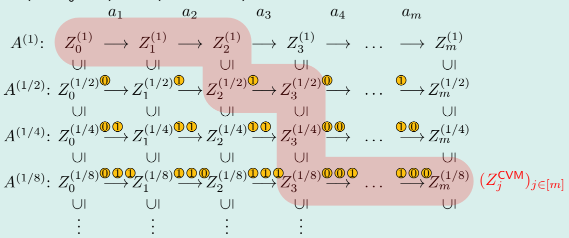
Algorithm update($(P, Z), a$):
- $Z \leftarrow Z \setminus \{a\}$
- **with probability** P **do**
  - $Z \leftarrow Z \cup \{a\}$
- **while** $|Z| \geq T$ **do** // shrink
  - $Z' \leftarrow \varnothing$
  - **for** $a \in Z$ **do**
    - **with probability** $1/2$ **do**
      - $Z' \leftarrow Z' \cup \{a\}$
  - $(Z, P) \leftarrow (Z', P/2)$
- **return** (P, Z)

Algorithm result((P, Z)):
- **return** $|Z|/P$

CVM behaves like LossyStore with dynamic p

Consider $A^{(p)} :=$ LossyStore(p) with states $Z_0^{(p)}, \ldots, Z_m^{(p)}$ for $p \in \{1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \ldots\}$.

Let $(P_0, Z_0^{(\mathrm{CVM})}), \ldots, (P_m, Z_m^{(\mathrm{CVM})})$ be the state of CVM.



Intuition: The path of CVM:
- $(x, y) \leftarrow (0, 0)$ // top left
- **for** $i = 1$ **to** m **do** // m updates
  - $x \leftarrow x + 1$ // go right
  - **while** $|Z_x^{(2^{-y})}| \geq T$ **do**
    - $y \leftarrow y + 1$ // go down

final state is $Z_m^{(2^{-y})}$

Coupling between executions of $A^{(p)}$ and CVM:

- $A^{(p/2)}$ uses coin tosses of $A^{(p)}$ and one more.
  "$A^{(p/2)}$ keeps half of what $A^{(p)}$ keeps."
- CVM uses coin tosses of $A^{(P)}$ to process elements.
- When shrinking, CVM inspects past coin tosses done by $A^{(P/2)}$. (the next unused coin for all $a \in Z$)

Effects of the coupling:

- $Z_j^{(\mathrm{CVM})} = Z_j^{(P_j)}$ for $j \in [m]$
- $\mathrm{result}^{(\mathrm{CVM})} = \mathrm{result}^{(P_m)}$
- $\mathrm{fail}^{(\mathrm{CVM})} = \mathrm{fail}^{(P_m)}$

Attempt III: Adjust lossiness dynamically

CVM, parameter T

Algorithm init:
- $Z \leftarrow \varnothing$
- $P \leftarrow 1$
- **return** (P, Z)

Algorithm update($(P, Z), a$):
- $Z \leftarrow Z \setminus \{a\}$
- **with probability** P **do**
  - $Z \leftarrow Z \cup \{a\}$
- **while** $|Z| \geq T$ **do** // shrink
  - $Z' \leftarrow \varnothing$
  - **for** $a \in Z$ **do**
    - **with probability** $1/2$ **do**
      - $Z' \leftarrow Z' \cup \{a\}$
  - $(Z, P) \leftarrow (Z', P/2)$
- **return** (P, Z)

Algorithm result((P, Z)):
- **return** $|Z|/P$

Lemma: Failure Probability and Space

With $T = \frac{18\log_2(2m/\delta)}{\varepsilon^2}$ we get $\Pr[\text{fail}^{\text{CVM}}] = \mathcal{O}(\delta)$ and $\text{space}^{\text{CVM}} = \mathcal{O}(\frac{\log(m/\delta)}{\varepsilon^2}\log n) + \lceil \log_2(\log_2(1/P_m)) \rceil$.

Analysis of CVM's failure probability (a bit sketchy)

- Recall: LossyStore$(p_\delta = \frac{3\log(2/\delta)}{\varepsilon^2 F_0})$ has failure probability $\leq \delta$. Assume p_δ is power of 2.
- Then $\Pr[\text{fail}^{(p_\delta)}] \leq \delta$, $\Pr[\text{fail}^{(2p_\delta)}] \leq \delta^2$, $\Pr[\text{fail}^{(4p_\delta)}] \leq \delta^4, \dots$.
- Therefore $\Pr[\text{fail}^{(1)}] + \dots + \Pr[\text{fail}^{(2p_\delta)}] + \Pr[\text{fail}^{(p_\delta)}] \leq \dots + \delta^8 + \delta^4 + \delta^2 + \delta = \mathcal{O}(\delta)$.
- $$\Pr[P_m < p_\delta] = \Pr[|Z_j^{(p_\delta)}| \geq T \text{ for some } j \in [m]] \leq m \cdot \Pr[|Z_m^{(p_\delta)}| \geq T]$$
  $$= m \cdot \Pr_{Z \sim \text{Bin}(F_0, p_\delta)}[Z \geq T] \overset{\Delta}{\leq} m \cdot 2^{-T} \leq m \cdot 2^{-\log(m/\delta)} = \delta.$$
  where Δ uses a Chernoff bound and $6\mathbb{E}[Z] = 6F_0 p_\delta = \dfrac{18\log_2(2/\delta)}{\varepsilon^2} \leq T$.
- $\text{fail}^{\text{CVM}} \Leftrightarrow \text{fail}^{(P_m)} \Rightarrow \left(P_m < p_\delta \vee \text{fail}^{(1)} \vee \text{fail}^{(1/2)} \vee \dots \vee \text{fail}^{(p_\delta)}\right)$

Finally: $\Pr[\text{fail}^{\text{CVM}}] \leq \Pr[P_m < p_\delta \vee \text{fail}^{(1)} \vee \text{fail}^{(1/2)} \vee \dots \vee \text{fail}^{(p_\delta)}] \overset{\text{UB}}{\leq} \delta + \mathcal{O}(\delta) = \mathcal{O}(\delta)$.

Conclusion

Streaming Algorithms

- Input read only once, from left to right.
- Goal: Use little space. (less than what is needed to store input stream)
- Motivation: Network actor wants to maintain statistic on traffic.

Morris$^+$ Algorithm for Counting the Stream Length

- approximation in space $\mathcal{O}(\frac{1}{\varepsilon^2\delta}\log\log m)$ // or $\mathcal{O}(\log\log m + \log\frac{1}{\delta\varepsilon})$ using Morris*?
  ($\varepsilon =$ relative error, $\delta =$ failure probability)
- deterministic algorithms need space $\lceil \log(1+m) \rceil$

CVM Algorithm for Counting *Distinct* Elements

- approximation in space $\mathcal{O}(\frac{1}{\varepsilon^2}\log(n)\log(m/\delta))$

Appendix: Possible Exam Questions I

- Definition of streaming algorithms:
  - What is the task of a streaming algorithm (with respect to a quantity $F = F(a_1, \ldots, a_m)$)?
  - What is the specific challenge for streaming algorithms?
- Streaming algorithms for $F_1 = m$:
  - What could be an application in which one would like to estimate F_1?
  - How much memory is needed if one simply counts? Can a deterministic algorithm do something smarter?
  - How does the LossyCounting algorithm work? Why does it not help us here?
  - How does Morris' algorithm work?
  - Prove that Morris' algorithm is unbiased.*
  - Prove that the memory usage of Morris is doubly logarithmic in m.
  - What other weakness did Morris' algorithm have, and how did we fix it?

Appendix: Possible Exam Questions II

Karlsruhe Institute of Technology

- Streaming algorithms for $F_0 = \{a_1, \ldots, a_m\}$:
  - What could be an application in which one would like to estimate F_0?
  - How much memory does the naive deterministic algorithm require? What can we achieve with CVM?
  - As an intermediate step, we formulated the LossyStore algorithm. How does it work?
  - How does the CVM algorithm work? How is it related to the LossyStore algorithm?
  - In the analysis of the error probability of CVM, we distinguished two types of problems. Which ones?*