CS-E4600
Mining data streams
slide set 7

Aristides Gionis
Department of Computer Science
Aalto University

reading assignment

- LRU book: chapter 4
- optional reading
  - paper by Alon, Matias, and Szegedy [Alon et al., 1999]
  - paper by Charikar, Chen, and Farach-Colton [Charikar et al., 2002]
  - paper by Cormode and Muthukrishnan [Cormode and Muthukrishnan, 2005]

data streams

- a data stream is a massive sequence of data
- too large to store (on disk, memory, cache, etc.)
- examples:
  - social media (e.g., twitter feed, foursquare checkins)
  - sensor networks (weather, radars, cameras, etc.)
  - network traffic (trajectories, source/destination pairs)
  - satellite data feed
- how to deal with such data?
- what are the issues?

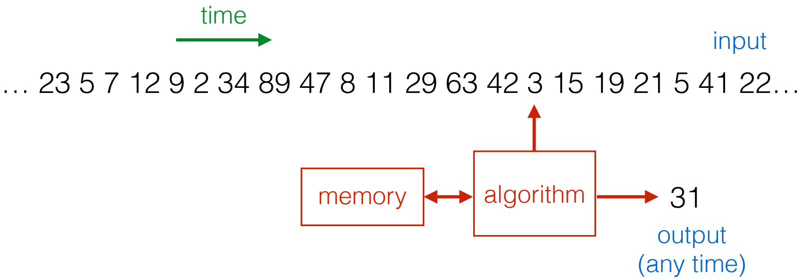
issues when working with data streams

- space
  - data size is very large
  - often not possible to store the whole dataset
  - inspect each data item, make some computations, do not store it, and never get to inspect it again
  - sometimes data is stored, but making one single pass takes a lot of time, especially when the data is stored on disk
  - can afford a small number of passes over the data
- time
  - data "flies by" at a high speed
  - computation time per data item needs to be small

data streams

- data items can be of complex types
  - documents (tweets, news articles)
  - images
  - geo-located time-series
  - ...
- to study basic algorithmic ideas we abstract away application-specific details
- consider the data stream as a sequence of numbers

data-stream model



data-stream model

- stream: m elements from universe of size n, e.g.,

$$\langle x_1, x_2, \ldots, x_m \rangle = 6, 1, 7, 4, 9, 1, 5, 1, 5, \ldots$$

- goal: compute a function over the elements of the stream, e.g., median, number of distinct elements, quantiles, ...
- constraints:
  1. limited working memory, sublinear in n and m e.g., $\mathcal{O}(\log n + \log m)$,
  2. access data sequentially
  3. limited number of passes, in some cases only one
  4. process each element quickly, e.g., $\mathcal{O}(1)$, $\mathcal{O}(\log n)$, etc.

warm up: computing some simple functions

- assume that a number can be stored in $\mathcal{O}(\log n)$ space
- `max`, `min` can be computed with $\mathcal{O}(\log n)$ space
- `sum`, `mean` (average) need $\mathcal{O}(\log n + \log m)$ space

$$\mu_X = \mathbb{E}[X] = \mathbb{E}[x_1, \ldots, x_m] = \frac{1}{m} \sum_{i=1}^{m} x_i$$

- what about variance?

$$\mathbb{V}ar[X] = \mathbb{V}ar[x_1, \ldots, x_m] = \mathbb{E}\left[(X - \mathbb{E}[X])^2\right]$$
$$= \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_X)^2$$

- two passes? one pass?

warm up: computing some simple functions

- assume that a number can be stored in $\mathcal{O}(\log n)$ space
- `max`, `min` can be computed with $\mathcal{O}(\log n)$ space
- `sum`, `mean` (average) need $\mathcal{O}(\log n + \log m)$ space

$$\mu_X = \mathbb{E}[X] = \mathbb{E}[x_1, \ldots, x_m] = \frac{1}{m}\sum_{i=1}^{m} x_i$$

- what about variance?

$$\mathbb{V}ar[X] = \mathbb{V}ar[x_1, \ldots, x_m] = \mathbb{E}\left[(X - \mathbb{E}[X])^2\right]$$
$$= \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_X)^2$$

- two passes? one pass?

warm up: computing some simple functions

- assume that a number can be stored in $\mathcal{O}(\log n)$ space
- `max`, `min` can be computed with $\mathcal{O}(\log n)$ space
- `sum`, `mean` (average) need $\mathcal{O}(\log n + \log m)$ space

$$\mu_X = \mathbb{E}[X] = \mathbb{E}[x_1, \ldots, x_m] = \frac{1}{m} \sum_{i=1}^{m} x_i$$

- what about variance?

$$\begin{aligned} \mathbb{V}ar[X] = \mathbb{V}ar[x_1, \ldots, x_m] &= \mathbb{E}\left[(X - \mathbb{E}[X])^2\right] \\ &= \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_X)^2 \end{aligned}$$

- two passes? one pass?

warm up: computing some simple functions

- assume that a number can be stored in $\mathcal{O}(\log n)$ space
- `max`, `min` can be computed with $\mathcal{O}(\log n)$ space
- `sum`, `mean` (average) need $\mathcal{O}(\log n + \log m)$ space

$$\mu_X = \mathbb{E}[X] = \mathbb{E}[x_1, \ldots, x_m] = \frac{1}{m}\sum_{i=1}^{m} x_i$$

- what about variance?

$$\begin{aligned}\mathbb{V}ar[X] = \mathbb{V}ar[x_1, \ldots, x_m] &= \mathbb{E}\left[(X - \mathbb{E}[X])^2\right] \\ &= \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_X)^2\end{aligned}$$

- two passes? one pass?

how to tackle massive data streams?

- a general and powerful technique: sampling
- idea:
  1. keep a random sample of the data stream
  2. perform the computation on the sample
  3. extrapolate
- example: compute the median of a data stream

  (how to extrapolate in this case?)
- but ... how to keep a random sample of a data stream?

how to tackle massive data streams?

- a general and powerful technique: sampling
- idea:
  1. keep a random sample of the data stream
  2. perform the computation on the sample
  3. extrapolate
- example: compute the median of a data stream

  (how to extrapolate in this case?)
- but . . . how to keep a random sample of a data stream?

reservoir sampling

- problem: take a uniform sample s from a stream of unknown length
- algorithm:
  - initially $s \leftarrow x_1$
  - on seeing the t-th element, $s \leftarrow x_t$ with probability $1/t$
- analysis:
  - what is the probability that $s = x_i$ at some time $t \geq i$?

$$\Pr[s = x_i] = \frac{1}{i} \cdot \left(1 - \frac{1}{i+1}\right) \cdot \ldots \cdot \left(1 - \frac{1}{t-1}\right) \cdot \left(1 - \frac{1}{t}\right)$$
$$= \frac{1}{i} \cdot \frac{i}{i+1} \cdot \ldots \cdot \frac{t-2}{t-1} \cdot \frac{t-1}{t} = \frac{1}{t}$$

- how much space? $\mathcal{O}(\log n)$
- to get k samples we need $\mathcal{O}(k \log n)$ bits

reservoir sampling

- problem: take a uniform sample s from a stream of unknown length
- algorithm:
  - initially $s \leftarrow x_1$
  - on seeing the t-th element, $s \leftarrow x_t$ with probability $1/t$
- analysis:
  - what is the probability that $s = x_i$ at some time $t \geq i$?

$$\begin{aligned} \Pr[s = x_i] &= \frac{1}{i} \cdot \left(1 - \frac{1}{i+1}\right) \cdot \ldots \cdot \left(1 - \frac{1}{t-1}\right) \cdot \left(1 - \frac{1}{t}\right) \\ &= \frac{1}{i} \cdot \frac{i}{i+1} \cdot \ldots \cdot \frac{t-2}{t-1} \cdot \frac{t-1}{t} = \frac{1}{t} \end{aligned}$$

- how much space? $\mathcal{O}(\log n)$
- to get k samples we need $\mathcal{O}(k \log n)$ bits

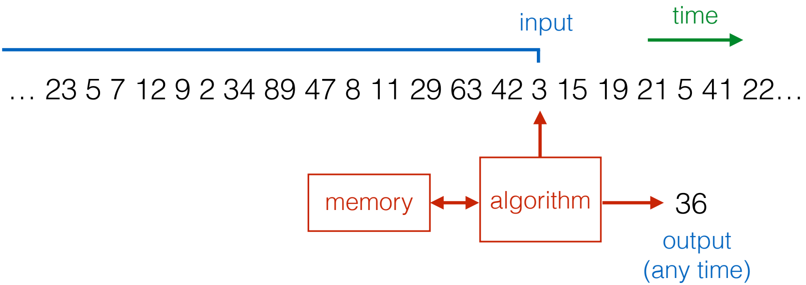
reservoir sampling

- problem: take a uniform sample s from a stream of unknown length
- algorithm:
  - initially $s \leftarrow x_1$
  - on seeing the t-th element, $s \leftarrow x_t$ with probability $1/t$
- analysis:
  - what is the probability that $s = x_i$ at some time $t \geq i$?

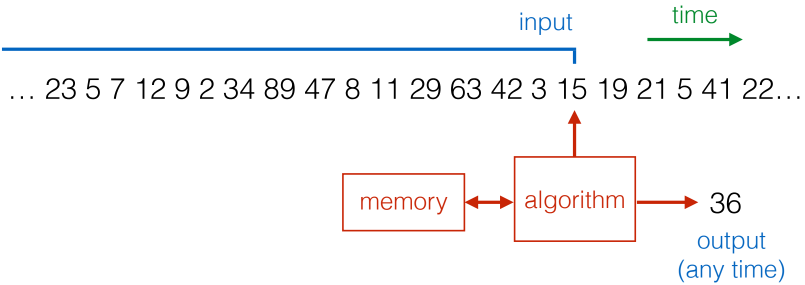
$$\begin{aligned}\Pr[s = x_i] &= \frac{1}{i} \cdot \left(1 - \frac{1}{i+1}\right) \cdot \ldots \cdot \left(1 - \frac{1}{t-1}\right) \cdot \left(1 - \frac{1}{t}\right) \\ &= \frac{1}{i} \cdot \frac{i}{i+1} \cdot \ldots \cdot \frac{t-2}{t-1} \cdot \frac{t-1}{t} = \frac{1}{t}\end{aligned}$$

- how much space? $\mathcal{O}(\log n)$
- to get k samples we need $\mathcal{O}(k \log n)$ bits

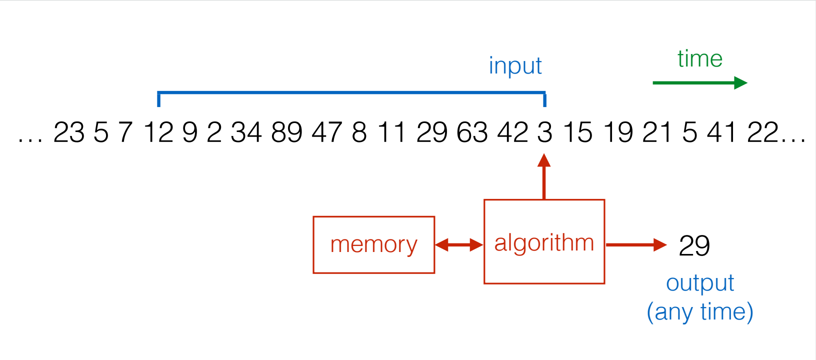
infinite data-stream model



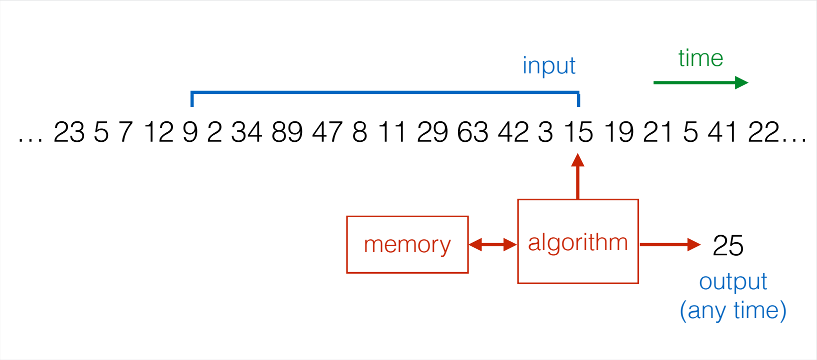
infinite data-stream model



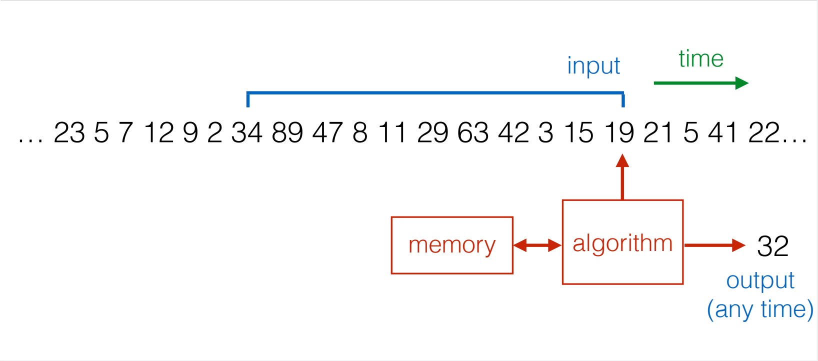
sliding-window data-stream model



sliding-window data-stream model



sliding-window data-stream model



sliding-window data-stream model

- does sliding-window model makes computation easier or harder?
- how to compute `sum`?
- how to keep a random sample?

- all computations can be done with $\mathcal{O}(w)$ space
- can we do better?

sliding-window data-stream model

- does sliding-window model makes computation easier or harder?
- how to compute `sum`?
- how to keep a random sample?

- all computations can be done with $\mathcal{O}(w)$ space
- can we do better?

sliding-window data-stream model

- does sliding-window model makes computation easier or harder?
- how to compute `sum`?
- how to keep a random sample?

- all computations can be done with $\mathcal{O}(w)$ space
- can we do better?

sliding-window data-stream model

- does sliding-window model makes computation easier or harder?
- how to compute `sum`?
- how to keep a random sample?

- all computations can be done with $\mathcal{O}(w)$ space
- can we do better?

priority sampling for sliding window

- maintain a uniform sample from the last w items
- reservoir sampling does not work in this model
- algorithm:
  1. for each x_i we pick a random value $v_i \in (0, 1)$
  2. for window $\langle x_{j-w+1}, \ldots, x_j \rangle$ return x_i with smallest v_i

  - to do this, maintain set of all elements in sliding window whose v value is minimal among all subsequent values

priority sampling for sliding window

- maintain a uniform sample from the last w items
- reservoir sampling does not work in this model
- algorithm:
  1. for each x_i we pick a random value $v_i \in (0, 1)$
  2. for window $\langle x_{j-w+1}, \ldots, x_j \rangle$ return x_i with smallest v_i

  - to do this, maintain set of all elements in sliding window whose v value is minimal among all subsequent values

priority sampling for sliding window

… 23 5 7 12 9 2 34 89 47 8 11 29 63 …

.64 .12 .31 .84 .27 .56 .91

priority sampling for sliding window

… 23 5 7 12 9 2 34 89 47 8 11 29 63 …

.64 .12 .31 .84 .27 .56 .91

priority sampling for sliding window

…	23	5	7	12	9	2	34	89	47	8	11	29	63 …
	.64	.12	.31	.84	.27	.56	.91						

priority sampling for sliding window

… 23 5 7 12 9 2 34 89 47 8 11 29 63 …

.64 .12 .31 .84 .27 .56 .91 .42

priority sampling for sliding window

… 23 5 7 12 9 2 34 89 47 8 11 29 63 …

.64 .12 .31 .84 .27 .56 .91 .42

priority sampling for sliding window

… 23 5 7 12 9 2 34 89 47 8 11 29 63 …

.64 .12 .31 .84 .27 .56 .91 .42 .73

priority sampling for sliding window

… 23 5 7 12 9 2 34 89 47 8 11 29 63 …

.64 .12 .31 .84 .27 .56 .91 .42 .73

priority sampling for sliding window

… 23 5 7 12 9 2 34 89 47 8 11 29 63 …

.64 .12 .31 .84 .27 .56 .91 .42 .73 .20

priority sampling for sliding window

… 23 5 7 12 9 2 34 89 47 8 11 29 63 …

.64 .12 .31 .84 .27 .56 .91 .42 .73 .20

priority sampling for sliding window

… 23 5 7 12 9 2 34 89 47 8 11 29 63 …

.64 .12 .31 .84 .27 .56 .91 .42 .73 .20

priority sampling for sliding window

- correctness 1: in any given window each item has equal chance to be selected as a random sample
- correctness 2: each minimal element x removed from memory has a smaller element y that comes after; x will expire before y, so x will never be needed again
- correctness 3: memory has always at least one element

priority sampling for sliding window

- correctness 1: in any given window each item has equal chance to be selected as a random sample
- correctness 2: each minimal element x removed from memory has a smaller element y that comes after; x will expire before y, so x will never be needed again
- correctness 3: memory has always at least one element

priority sampling for sliding window

- correctness 1: in any given window each item has equal chance to be selected as a random sample
- correctness 2: each minimal element x removed from memory has a smaller element y that comes after; x will expire before y, so x will never be needed again
- correctness 3: memory has always at least one element

priority sampling for sliding window

- correctness 1: in any given window each item has equal chance to be selected as a random sample
- correctness 2: each minimal element x removed from memory has a smaller element y that comes after; x will expire before y, so x will never be needed again
- correctness 3: memory has always at least one element

priority sampling for sliding window

- space efficiency: how many minimal elements do we expect at any given point?
- expected number of minimal elements is $\mathcal{O}(\log w)$
- so, expected space requirement is $\mathcal{O}(\log w \log n)$

- time efficiency: maintaining list of minimal elements requires $\mathcal{O}(\log w)$ time

priority sampling for sliding window

- space efficiency: how many minimal elements do we expect at any given point?
- expected number of minimal elements is $\mathcal{O}(\log w)$
- so, expected space requirement is $\mathcal{O}(\log w \log n)$

- time efficiency: maintaining list of minimal elements requires $\mathcal{O}(\log w)$ time

priority sampling for sliding window

- space efficiency: how many minimal elements do we expect at any given point?
- expected number of minimal elements is $\mathcal{O}(\log w)$
- so, expected space requirement is $\mathcal{O}(\log w \log n)$
- time efficiency: maintaining list of minimal elements requires $\mathcal{O}(\log w)$ time

mining data streams

- what are real-world applications?
- imagine monitoring a social feed stream
  - a stream of hashtags in twitter
  - what are interesting questions to ask?
  - do data stream considerations (space/time) really matter?

mining data streams

- what are real-world applications?
- imagine monitoring a social feed stream
  - a stream of hashtags in twitter
  - what are interesting questions to ask?
  - do data stream considerations (space/time) really matter?

how to tackle massive data streams?

- a general and powerful technique: sketching
- general idea:
- apply a linear projection that takes high-dimensional data to a smaller dimensional space
- post-process lower dimensional image to estimate the quantities of interest

computing statistics on data streams

- $X = (x_1, x_2, \ldots, x_m)$ a sequence of elements
- each x_i is a member of the set $N = \{1, \ldots, n\}$
- $m_i = |\{j : x_j = i\}|$ the number of occurrences of i
- define the k-th frequency moment

$$F_k = \sum_{i=1}^{n} m_i^k$$

- F_0 is the number of distinct elements
- F_1 is the length of the sequence
- F_2 is the second moment: index of homogeneity, size of self-join, and other applications
- F_∞^* frequency of most frequent element

computing statistics on data streams

- $X = (x_1, x_2, \ldots, x_m)$ a sequence of elements
- each x_i is a member of the set $N = \{1, \ldots, n\}$
- $m_i = |\{j : x_j = i\}|$ the number of occurrences of i
- define the k-th frequency moment

$$F_k = \sum_{i=1}^{n} m_i^k$$

- F_0 is the number of distinct elements
- F_1 is the length of the sequence
- F_2 is the second moment: index of homogeneity, size of self-join, and other applications
- F_∞^* frequency of most frequent element

computing statistics on data streams

- $X = (x_1, x_2, \ldots, x_m)$ a sequence of elements
- each x_i is a member of the set $N = \{1, \ldots, n\}$
- $m_i = |\{j : x_j = i\}|$ the number of occurrences of i
- define the k-th frequency moment

$$F_k = \sum_{i=1}^{n} m_i^k$$

- F_0 is the number of distinct elements
- F_1 is the length of the sequence
- F_2 is the second moment: index of homogeneity, size of self-join, and other applications
- F_∞^* frequency of most frequent element

computing statistics on data streams

- $X = (x_1, x_2, \ldots, x_m)$ a sequence of elements
- each x_i is a member of the set $N = \{1, \ldots, n\}$
- $m_i = |\{j \,:\, x_j = i\}|$ the number of occurrences of i
- define the k-th frequency moment

$$F_k = \sum_{i=1}^{n} m_i^k$$

- F_0 is the number of distinct elements
- F_1 is the length of the sequence
- F_2 is the second moment: index of homogeneity, size of self-join, and other applications
- F_∞^* frequency of most frequent element

computing statistics on data streams

- $X = (x_1, x_2, \ldots, x_m)$ a sequence of elements
- each x_i is a member of the set $N = \{1, \ldots, n\}$
- $m_i = |\{j \,:\, x_j = i\}|$ the number of occurrences of i
- define the k-th frequency moment

$$F_k = \sum_{i=1}^{n} m_i^k$$

- F_0 is the number of distinct elements
- F_1 is the length of the sequence
- F_2 is the second moment: index of homogeneity, size of self-join, and other applications
- F_∞^* frequency of most frequent element

computing statistics on data streams

- how much space I need to compute the frequency moments in a straighforward manner?
- how to compute the frequency moments using less than $O(n \log m)$ space?
- problem studied by Alon, Matias, Szegedy [Alon et al., 1999]
- sketching: create a sketch that takes much less space and gives an estimation of F_k

computing statistics on data streams

- how much space I need to compute the frequency moments in a straighforward manner?
- how to compute the frequency moments using less than $O(n \log m)$ space?
- problem studied by Alon, Matias, Szegedy [Alon et al., 1999]
- sketching: create a sketch that takes much less space and gives an estimation of F_k

computing statistics on data streams

- how much space I need to compute the frequency moments in a straighforward manner?
- how to compute the frequency moments using less than $O(n \log m)$ space?
- problem studied by Alon, Matias, Szegedy [Alon et al., 1999]
- sketching: create a sketch that takes much less space and gives an estimation of F_k

estimating the number of distinct values (F_0)

[Flajolet and Martin, 1985]

- consider a bit vector $\mathbf{b}$ with $O(\log n)$ bits
- initialize $\mathbf{b}$ to $[0, \ldots, 0]$
- consider a hash function f that maps each item x to the j-th bit of the bit-vector $\mathbf{b}$ with probability $1/2^j$
- for each item x_i in the data stream

  set the bit $j = f(x_i)$ of $\mathbf{b}$ equal to 1

  (important: bits are set deterministically for each x_i)
- let R be the index of the largest bit set
- return $Y = 2^R$

estimating the number of distinct values (F_0)

[Flajolet and Martin, 1985]

intuition:

- the j-th bit of $\mathbf{b}$ is set with probability $1/2^j$
- e.g., after seeing 32 distinct elements
  - the bits $1, 2, 3, 4, 5$ are most likely set
  - the bits $6, 7, \ldots$ are most likely not set
- i.e., we expect the bit vector to be 0000011111, and thus the estimate is 32

estimating number of distinct values (F_0)

Theorem. For every $c > 2$, the algorithm computes a number Y using $\mathcal{O}(\log n)$ memory bits, such that the probability that the ratio between Y and F_0 is not between $1/c$ and c is at most $2/c$.

Theorem proven in [Alon et al., 1999]

estimating F_2

- $X = (x_1, x_2, \ldots, x_m)$ a sequence of elements
- each x_i is a member of the set $N = \{1, \ldots, n\}$
- $m_i = |\{j : x_j = i\}|$ the number of occurrences of i
- $F_k = \sum_{i=1}^{n} m_i^k$

- algorithm:
- hash each $i \in \{1, \ldots, n\}$ to a random $\epsilon_i \in \{-1, +1\}$
- maintain sketch $Z = \sum_i \epsilon_i m_i$

  just need space $\mathcal{O}(\log n + \log m)$
- take $X = Z^2$
- return the average Y of k such estimates $X_1, \ldots, X_k$
- $Y = \frac{1}{k} \sum_{j=1}^{k} X_j$ where $k = \frac{16}{\lambda^2}$

estimating F_2

- $X = (x_1, x_2, \ldots, x_m)$ a sequence of elements
- each x_i is a member of the set $N = \{1, \ldots, n\}$
- $m_i = |\{j : x_j = i\}|$ the number of occurrences of i
- $F_k = \sum_{i=1}^{n} m_i^k$

- algorithm:
- hash each $i \in \{1, \ldots, n\}$ to a random $\epsilon_i \in \{-1, +1\}$
- maintain sketch $Z = \sum_i \epsilon_i m_i$

  just need space $\mathcal{O}(\log n + \log m)$
- take $X = Z^2$
- return the average Y of k such estimates $X_1, \ldots, X_k$
- $Y = \frac{1}{k} \sum_{j=1}^{k} X_j$ where $k = \frac{16}{\lambda^2}$

expectation of the estimate is correct

$$
\begin{aligned}
\mathbb{E}[X] &= \mathbb{E}\left[Z^2\right] \\
&= \mathbb{E}\left[\left(\sum_{i=1}^{n} \epsilon_i m_i\right)^2\right] \\
&= \sum_{i=1}^{n} m_i^2 \mathbb{E}\left[\epsilon_i^2\right] + 2\sum_{i<j} m_i m_j \mathbb{E}\left[\epsilon_i\right] \mathbb{E}\left[\epsilon_j\right] \\
&= \sum_{i=1}^{n} m_i^2 = F_2
\end{aligned}
$$

accuracy of the estimate

easy to show

$$\mathbb{E}\left[X^2\right] = \sum_{i=1}^{n} m_i^4 + 6 \sum_{i<j} m_i^2 m_j^2$$

which gives

$$\mathbb{V}ar\left[X\right] = \mathbb{E}\left[X^2\right] - \mathbb{E}\left[X\right]^2 = 4 \sum_{i<j} m_i^2 m_j^2 \leq 2F_2^2$$

and by Chebyshev's inequality

$$\Pr[|Y - F_2| \geq \lambda F_2] \leq \frac{\mathbb{V}ar\left[Y\right]}{\lambda^2 F_2^2} = \frac{\mathbb{V}ar\left[X\right]/k}{\lambda^2 F_2^2} \leq \frac{2F_2^2/k}{\lambda^2 F_2^2} = \frac{2}{k\lambda^2} = \frac{1}{8}$$

accuracy of the estimate

easy to show

$$\mathbb{E}\left[X^2\right] = \sum_{i=1}^{n} m_i^4 + 6\sum_{i<j} m_i^2 m_j^2$$

which gives

$$\mathbb{V}ar\left[X\right] = \mathbb{E}\left[X^2\right] - \mathbb{E}\left[X\right]^2 = 4\sum_{i<j} m_i^2 m_j^2 \leq 2F_2^2$$

and by Chebyshev's inequality

$$\Pr[|Y - F_2| \geq \lambda F_2] \leq \frac{\mathbb{V}ar\left[Y\right]}{\lambda^2 F_2^2} = \frac{\mathbb{V}ar\left[X\right]/k}{\lambda^2 F_2^2} \leq \frac{2F_2^2/k}{\lambda^2 F_2^2} = \frac{2}{k\lambda^2} = \frac{1}{8}$$

accuracy of the estimate

easy to show

$$\mathbb{E}\left[X^2\right] = \sum_{i=1}^{n} m_i^4 + 6 \sum_{i<j} m_i^2 m_j^2$$

which gives

$$\mathbb{V}ar\left[X\right] = \mathbb{E}\left[X^2\right] - \mathbb{E}\left[X\right]^2 = 4 \sum_{i<j} m_i^2 m_j^2 \leq 2F_2^2$$

and by Chebyshev's inequality

$$\Pr[|Y - F_2| \geq \lambda F_2] \leq \frac{\mathbb{V}ar\left[Y\right]}{\lambda^2 F_2^2} = \frac{\mathbb{V}ar\left[X\right]/k}{\lambda^2 F_2^2} \leq \frac{2F_2^2/k}{\lambda^2 F_2^2} = \frac{2}{k\lambda^2} = \frac{1}{8}$$

estimate of F_2 : summing up

Theorem. Let $X_1, \ldots, X_k$ be AMS sketches, with $k = \frac{16}{\lambda^2}$, and Y be their average $Y = \frac{1}{k}\sum_{j=1}^{k} X_j$.

Then, Y is an unbiased estimator of F_2, and the quality of approximation is given by

$$\Pr[|Y - F_2| \geq \lambda F_2] \leq \frac{1}{8}$$

references I

Alon, N., Matias, Y., and Szegedy, M. (1999).
The space complexity of approximating the frequency moments.
J. Comput. Syst. Sci., 58(1):137–147.

Charikar, M., Chen, K., and Farach-Colton, M. (2002).
Finding frequent items in data streams.
In *International Colloquium on Automata, Languages, and Programming*, pages 693–703.

Cormode, G. and Muthukrishnan, S. (2005).
An improved data stream summary: the count-min sketch and its applications.
Journal of Algorithms, 55(1):58–75.

Flajolet, P. and Martin, N. G. (1985).
Probabilistic counting algorithms for data base applications.
Journal of Computer and System Sciences, 31(2):182–209.