3 Mar 2025 — Misra-Gries and Count-Min Sketch

Announcement: Quiz 4 is graded. Regrades until Sunday.

PSet 3 to be released soon. (Wed?)

Due late next week (Thurs / Fri)

Recap. Finding frequent elements in a stream with false positives allowed.

MISRA-GRIES. An algorithm using $O(k \cdot (b + \log n))$ bits of storage for any specified k. $\equiv O\left(\frac{kb}{\log n} + k\right)$ storage space.

Assumption: Stream consists of n elements of $\{0,1\}^b$.

Guarantee. Outputs at most k elements of the stream. Every element occurring $> \frac{n}{k+1}$ times will be reported among the output.

Algorithm. Initialize array of k ordered pairs, each initially $(\perp, \perp)$.

for $i = 1, \ldots, n$.
  read x_i from stream.
  if buffer contains ordered pair (x_i, c)
    increment counter to $(x_i, c+1)$
  elif buffer contains $(\perp, \perp)$
    replace with $(x_i, 1)$
  else

decrement **every** counter in buffer,
replace any $(x, 0)$ with $(\perp, \perp)$.

E.g. $k = 3$

Bobby	Jon	Éva
6	1	4

Éva

Bobby	Jon	Éva
6	1	5

Eshan

Bobby	$\perp$	Éva
5	$\perp$	4

Éva

Bobby	$\perp$	Éva
5	$\perp$	5

Eshan

Bobby	Eshan	Eva
5	1	5

Analysis. We need to show, if x occurs more than $\frac{n}{k+1}$ times, that it will be present in the buffer when the alg. terminates.

As the algorithm runs, the buffer keeps track of every element that hasn't been grouped into a $(k+1)$-tuple of distinct elements.

When the decrementing-counter operation is performed, a new group of $k+1$ distinct elements is formed and accounted for.

If x occurs $> \frac{n}{k+1}$ times in the stream, each group of $k+1$ distinct elements contains ≤ 1 copy of x.

$\#$ groups $\leq \frac{n}{k+1}$.

$\Longrightarrow$ at least one copy of x was never placed in one of the groups.

$\Longrightarrow$ the buffer still contains x

$\Longrightarrow$ the algorithm's output contains x.

Data Sketching — Streaming meets data structures.

A sketching algorithm runs in two stages.

PREPROCESSING: Read a stream of data. Build a space efficient representation ("sketch") in memory.

QUERY: Receive queries about the data and attempt to answer them with *approximately* correct answers.

Sketching element frequencies.

For data stream $\underline{x} = x_1, x_2, \ldots, x_n \in \{0,1\}^b$

the frequency vector $\vec{f}$ has coordinates indexed by elements of $\{0,1\}^b$,

$f[x]$ = # of times x occurs in the stream.

Frequency query: given $x \in \{0,1\}^b$ return $f[x]$.

Lossless representations of the stream:

(1) $x_1, \ldots, x_n$: $n \cdot b$ bits

(2) $\vec{f}$: $2^b \cdot \log(n)$ bits

We are interested in lossy representations that use poly$\left(b, \log n, \frac{1}{\varepsilon}, \log\left(\frac{1}{\delta}\right)\right)$ bits and enable (ε, δ)-PAC answers to frequency queries.

Plan: use hashing!

With a single hash function
$$h : \{0,1\}^b \longrightarrow [B]$$
we could store a different frequency vector $g \in \mathbb{N}^B$,
$$g[b] = \#\{i \mid h(x_i) = b\}.$$

Properties of g:

- $g[h(x)] \geq f[x]$.
  (could be strictly greater due to hash collisions!)

- Space requirement for computing and storing g
  - store representation of h.
    $O(b \cdot \log(B))$ bits

$\mathbb{F}_p$ elements require $\log(B)$ bits
for $B \le p \le 2B$
Inner product hashing with
b-dim vectors uses
$b+1$ coefficients in $\mathbb{F}_p$.

- storing g itself.

$$g \in \{0, 1, \ldots, n\}^B$$

$$O(B \cdot (\log n)) \text{ bits.}$$

Lemma. If h is sampled randomly from a 2-universal hash family

$$\forall x \quad \Pr\left(g[h(x)] - f[x] > \frac{2n}{B}\right) < \frac{1}{2}.$$

Proof. $g[h(x)] - f[x]$ counts

$$\# \{ i \mid x_i \neq x \text{ but } h(x_i) = h(x) \}.$$

$$\mathbb{E}\left[\# \{ i \mid x_i \neq x \text{ but } h(x_i) = h(x) \} \right]$$

$$= \sum_{i=1}^{n} \Pr\left(x_i \neq x \text{ but } h(x_i) = h(x) \right)$$

$$\leq n \cdot \frac{1}{B}$$

(Markov)

$$\underline{\Pr}\left(g[h[x]] - F[x] > \frac{2n}{B} \right) < \frac{1}{2}$$

random sampling of h.

We will set $B = \left\lceil \frac{2}{\varepsilon} \right\rceil$ so

$$\frac{2n}{B} \leq \varepsilon n.$$

Then we have an (ε, δ)-PAC algorithm with $\delta = \frac{1}{2}$ (bad!) and ε arbitrarily close to 0 (good!) and space complexity

$$O\left(b \cdot \log\left(\frac{1}{\varepsilon}\right) + \frac{1}{\varepsilon} \cdot \log(n) \right)$$ bits. (good!)

<u>Count-min sketch</u>: Use $t = \log\left(\frac{1}{\delta}\right)$ hash functions instead of one. Sample hash functions <u>independently</u> from a 2-universal family.

Data structure encodes:

- hash functions $h_1, \ldots, h_t$

$$O(b \cdot t \cdot \log B) \text{ bits}$$

- 2-D array of counters,

$$G[b, j] = \#\{ i \mid h_j(x_i) = b \}.$$

$$O(B \cdot t \cdot \log n) \text{ bits}$$

Preprocessing

Initially sample $h_1, \ldots, h_t$.

Initialize $G[b, j] = 0 \quad \forall b, j$.

for $i = 1, \ldots, n$:

for $j = 1, \ldots, t$:

compute $b = h_j(x_i)$

increment $G[b, j]$

Query(x): Return $\min_j G[h_j(x), j]$.

One-sided error. $\forall x \in \{0,1\}^b \quad \forall j \in [t]$

$$G[h_j(x), j] \geq f[x].$$

$$\#\{i \mid h_j(x_i) = h_j(x)\}$$

$$\geq \quad \#\{i \mid x_i = x\}.$$

Fact 1. Query(x) returns a value $\geq f[x]$.

Fact 2. $\forall x \; \Pr\left(\text{Query}[x] - f[x] > \frac{2n}{B}\right) < 2^{-t}.$

We chose $B = \lceil \frac{2}{\varepsilon} \rceil$ so $\frac{2n}{B} \leq \varepsilon n$.

$t = \log(\frac{1}{\delta})$ so $2^{-t} = \delta$.

Translation: $\forall x \; \Pr(\text{Query}(x) - f[x] > \varepsilon n) < \delta.$

(ε, δ) — PAC.

(GOOD!)

Space requirement:

$$O\left(b \cdot \log\left(\frac{1}{\delta}\right) \cdot \log\left(\frac{1}{\varepsilon}\right) + \frac{1}{\varepsilon} \cdot \log\left(\frac{1}{\delta}\right) \cdot \log n\right)$$

(good)