Variants of the Streaming Model

Notation: number of stream updates (aka stream length): m

data universe size: n

stream itself: σ

- Vanilla stream model aka insert-only, unit-weight update model

  $\sigma = \langle a_1, \dots a_m \rangle$, each $a_i \in [n] := \{1, \dots n\}$

- cash register model aka insert-only model:

  $\sigma = \langle (a_1, \delta_1), \dots, (a_m, \delta_m) \rangle$, each $a_i \in [n]$, each $\delta_i \in \mathbb{Z}_+$

  interpreted as δ_1 "copies" of a_1

  Frequencies and frequency vector:

  Let $f_j := \sum_{i: a_i = j} \delta_i$. "frequency of item j"

  Let $f := (f_1, \dots f_n)$ "frequency vector of σ".

  Let $M := \sum_i \delta_i$ be the L_1-norm of the frequency vector

- Turnstile model: Same as cash register, but $\delta_i \in \mathbb{Z}$
- Strict turnstile model: Same as turnstile model, but promised that at end of stream, $\forall j$ $f_j \geq 0$.

~~Approximation Guarantees~~

~~Say we want to compute some re~~

Deterministic Algorithms For Approximating Frequencies in Insert-Only Streams.

Goal: Compute a summary of σ so that, for any $j \in [n]$, one can use the summary to output an estimate $\hat{f}_j$ of f_j such that
$|\hat{f}_j - f_j| \leq \epsilon \cdot M$.

A question of the form "What is f_j" is also called a point query

An algorithm for the vanilla stream model (where $M = m$).

Mishra-Gries, 1982:
- Algorithm maintains $K \approx \frac{1}{\epsilon}$ counters [initially unassigned] with
- Each counter is assigned a universe item $j \in [n]$ that's appeared in the stream.
- On stream update a_i, algorithm looks to see if a_i is assigned a counter.
  - If so, the count is incremented.
  - If not, and there are any unassigned counters, a_i is assigned a counter, and the count is incremented to 1.
  - Otherwise, all counters are decremented and any counters that become 0 are marked unassigned.
- When asked to provide an estimate $\hat{f}_j$ for f_j, the algorithm returns 0 if j is not assigned a counter, and returns the value of the counter otherwise.
- Claim: For each $j \in [n]$, $0 \leq f_j - \hat{f}_j \leq \frac{m}{K+1} \leq \epsilon m$.

Proof: It is obvious that $0 \le f_j - \hat{f}_j$ for all j.
Let us prove that $f_j - \hat{f}_j \le \frac{m}{(k+1)}$

Claim: The total number of decrement operations over the course of the algorithm is at most $\frac{m}{k+1}$.

Proof: If there are d decrement operations then since each decrement operation affects all k counters, the sum of the counter values at the end of the algorithm is $m - d\cdot(k+1)$. Since no counter value is ever negative, it must be that $m - d\cdot(k+1) \ge 0 \Rightarrow d \le \frac{m}{(k+1)}$.
Since the error in any estimate is at most the total number of decrement operations we are done

Remarks: • Space usage of algorithm is $O(k \cdot \log(n+m))$ bits,

$$\parallel$$

$$O\left(\frac{\log(n+m)}{\epsilon}\right),$$

This is ~~[illegible]~~ optimal up to a factor of $\log(n+m)$.
(For any subset $S \subseteq [n]$ of size $\frac{1}{\epsilon}$, there is a stream for which $f_j \ge \epsilon \cdot m \Leftrightarrow j \in S$. Hence, any sketch achieving our goal can be used to identify an arbitrary subset S of $[n]$ s.t. $|S| = \frac{1}{\epsilon}$

This "clearly" takes $\log\left(\binom{n}{1/\epsilon}\right) \approx \log\left((\epsilon n e)^{1/\epsilon}\right) = O\left(\frac{1}{\epsilon}\log(\epsilon n)\right)$ bits

$$\left(\frac{ne}{k}\right)^k \le \binom{n}{k} \le n^k$$

• The error guarantee $0 \le f_i - \hat{f}_i \le \frac{m}{k+1}$ can be improved to $0 \le f_i - \hat{f}_i \le \frac{F_1^{res(c)}}{k+1-c}$ for any c ← sum of frequencies of all but top-k most frequent items

This will be on the homework.

- Metwally et al. 2005
  Space Saving is same as Misra-Gries, except if a_i is not assigned a counter, the smallest counter is reassigned to a_i and incremented. Its estimates are always over estimates, not underestimates.

- These were treated as distinct algorithms for years, but actually they are isomorphic in the sense that given a "MG-sketch" with K counters one can compute the ~~Space Saving~~ estimates that would be returned by the Space Saving when run with K+1 counters, and vice versa.

  Homework.

Implementation ~~details~~ of Misra-Gries:

- Store all ≤k assigned counters in a hash table
  Keys are assigned items
  values are counts
- Assume all hash table operations (lookup, insert, delete) take $O(1)$ time (amortized)
- Assume enumerating all key-value pairs can be done in $O(k)$ time

Then every stream update can be processed in $O(1)$ amortized time, since each stream update requires

1) A hash table lookup ← $O(1)$ time
2) Either changing a key's value ← $O(1)$ time or
   - inserting ← $O(1)$ time a new assigned counter with value 1 or
   - A decrement operation (requires enumerating all up to K of them) K (key, value) pairs and deleting

     $O(k)$ time, and it only happens a total of at most $\frac{m}{k}$ times.

What about weighted stream updates?

Possibilities: • Process an update (a_i, δ_i) as δ_i unit-weight updates

• Berinde et al. 2010: when processing (a_i, δ_i):

  • if a_i is assigned a counter, increment it by δ_i
  • if not and there is an unassigned counter, assign it to i & set its value to δ_i
  • Otherwise decrement all counters by the <u>minimum</u> counter value. Easy to show $0 \le f_i - \hat{f}_i \le \frac{m}{k+1}$

Shortcoming: doesn't run in $O(1)$ amortized time per stream update. Why?

Proposal (upcoming work): decrement by the <u>median</u> counter value. Then this <u>does</u> run in $\boxed{\text{amortized } O(1)}$ time per stream update. Can also prove

$$0 \le f_i - \hat{f}_i \le \frac{2m}{k+2}$$ with this approach.

uses an algorithm for finding the median of k items in O(k) time (sorting takes O(k log k) time)

Really only approximate medians are necessary. We will see easy sampling-based algorithms for computing approximate medians in a few lectures

If time, talk about heavy hitters, <u>itemsets</u>.