STREAMING ALGORITHMS

Space! → the key resource today.

stream of data : $a_1, a_2, \ldots, a_t, \ldots$

one datapoint arrives at each time t

Goal: Maintain some function of a_i's seen up to time t

Ex: a_i's are integers.

Maintain the sum.

Idea: maintain a "sum" variable.

Add a_t at time step t to the current sum.

Alg:

sum $\leftarrow 0$

at time t,

sum $\leftarrow$ sum $+ a_t$

Let's analyze how good this algorithm is.

Key resource: Space!

Assume: each a_i is an integer in $[1, \ldots, N]$

Then each a_i can be written in

$$\lceil \log_2 N \rceil \text{ bits.}$$

How large a space do I need to store "Sum"?

All integers are $[1, \ldots, N]$

Need $\lceil \log_2 N \rceil$ bits.

At time T, sum $\leq T \cdot N$

$$\lceil \log_2 TN \rceil \leq \lceil \log_2 T \rceil + \lceil \log_2 N \rceil$$

Space Usage: $O(\log_2 T + \log_2 N)$

Ex 2: "Maximum"

alg: max $\leftarrow 0$
at time t
if max $< a_t$, max $\leftarrow a_t$
else discard a_t

Space usage? $\lceil \log_2 N \rceil$

Median ??? But think about it.

Since we need $\lceil \log_2 N \rceil$ space to even write down an element in the stream, $O(\log_2 N)$ is the "best" case space usage.

A trivial upper bound would be $O(T \cdot \log_2 N)$ (store the entire stream!)

Our goal: find algorithms that do

$$\ll T \log_2 n$$

& in fact are close to $O(\log_2 T + \log_2 N)$

"Heavy Hitters" Fix ε.

Given $a_1, \ldots, a_t \in \Sigma$

At all times t, maintain a list of all elements that have occurred $> \varepsilon t$ times.

Note: do not need a list of size $\geq \left\lceil \frac{1}{\varepsilon} \right\rceil - 1$.

Why? Exercise: prove that there cannot be more than $\left\lceil \frac{1}{\varepsilon} \right\rceil - 1$ heavy hitters.

Key Principles

1) Approximation is okay & important.

2) Hashing / randomness is crucial.

Heavy Hitters

Def: $\text{Count}_t(e) = \left|\{i \in \{1, \ldots, t\} \mid a_t = e\}\right|$

alphabet of stream

Def: Element $e \in \Sigma$ is an ε-heavy hitter at time t if $\text{Count}_t(e) > \varepsilon \cdot t$.

Relaxation of the guarantee

False positives ✓

False negatives ✗

"think of this as the relaxation/approximation we chose"

So, if there is a heavy hitter, it will be output. But all outputs need not be heavy hitters.

Finding a Majority element ($\equiv \frac{1}{2}$-heavy hitter)

Alg:

memory = $\perp$

Counter = 0

When element a_t arrives

if (Counter = 0),

set memory = a_t, Counter = 1

else

if a_t == memory,

Counter ← counter + 1

else

Counter ← counter − 1

discard a_t

Creturn the element in the memory)

Analysis:

Claim 1: # $\frac{1}{2}$-heavy hitters ≤ 1 at all times T.

Pf: if not, there are 2 distinct elements that each occur $> \frac{1}{2} \cdot T$ times. But since total stream length is $\leq T$. Contradiction. □

Claim 2: At all times $T > 0$, memory contains the majority element if it exists.

Pf: (Note: we make no claims if there is no Majority element)

Key: When we discard an element a_t from the memory, we also throw away another (different) element as well.

Decrementing the counter ≡ throwing away one copy of element in memory

So each time we discard the true majority we must have thrown away another diff element

If majority element is not in the memory, then # elements thrown away is $> T/2 + T/2 > T$. Contradiction

□

Space Usage: $O(\log_2 N + \log_2 T)$ → counter

↙

memory = single element

Let's now go figure out ε-heavy hitters.

Algo

Let $k = \lceil \frac{1}{\varepsilon} \rceil - 1$

$T[1, \ldots, k] = \perp^k$

$C[1, \ldots, k] = 0^k$

When element a_t arrives:

if $\exists\ j \in \{1, \ldots, k\}$ s.t. $a_t = T[j]$,
then $C[j]$++.

else if some counter $C[j] == 0$,
then $T[j] \leftarrow a_t$
$C[j] \leftarrow 1$

else: decrement all counters by 1
(and discard a_t).

Ex: check why this algo is same as before if $\varepsilon = 1/2$.

Def: $est_t(e) = \begin{cases} C[j] \text{ if } e = T[j] \\ 0 \quad \text{o/w} \end{cases}$

Lemma 1: "our estimate & the true count are somewhat close"

$$0 \leq count_t(e) - est_t(e) \leq \frac{t}{k+1} \leq \varepsilon \cdot t$$

Corollary:

Therefore if $count_t(e) > \varepsilon t$

then $est_t(e) > 0$

Proof: the sum of the counters cannot be more than t.

Whenever I decrement, it goes down by $k+1$.

Therefore no counter is dec by more

than $\frac{t}{k+1}$.

Distinct Elements

Input stream: $a_1, a_2, \ldots, a_t$

Goal: Keep a count of number of distinct elements seen so far.

Not hard to show that exact count requires memory $\cap T$.

Our goal: Keep a est-count s.t.

$$\frac{1}{C} \cdot \text{true-count} \leq \underset{\text{Count}}{\text{est}} \leq C \cdot \text{true-Count}$$

for some constant $C > 0$.

"Constant factor approx"

It turns out the every deterministic algorithm provably needs $\Omega(T)$ memory even for the approx. guarantee

But, there is a simple randomized algorithm!

Idea: Hashing.

the $\min_{i \le t} h(a_i)$

$\min = 1$

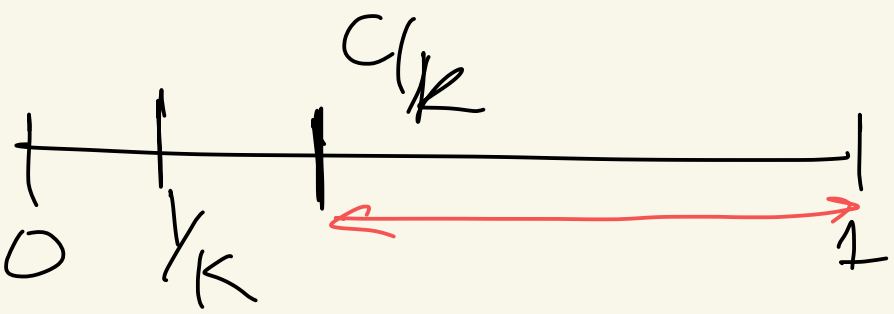
at time t,

compute $h(a_t)$

Switch if $h(a_t)$ is tinier than $\min$

Analysis of Idealized Algorithm

Lemma 1: Let k be the number of distinct elements. The probability that the min hash value is larger then $\frac{c}{k}$ is at most $1/100$ (for large enough constant $c > 0$).



Pf:

Chance that all k numbers are assigned in $[\frac{c}{k}, 1)$ is $\leq (1 - \frac{c}{k})^k$

$= e^{-c}$

Lemma 2: