Lecture 22 (November 18)

Streaming Algorithms

In today's lecture, we will look at streaming algorithms where randomness is often useful for designing algorithms

A data stream is an extremely long sequence of items from a universe that can only be read once in order

$a_1, a_2, a_3, \ldots\ldots, a_m$ where each $a_m \in \mathcal{U}$ where $\mathcal{U}$ is a set of n items

E.g. Packets passing through a network router
Sequence of google searches
NY Stock exchange trades

Standard algorithms are not suitable for computation because there is simply too much data to store and it arrives too quickly for complex computations

Ideally, one wants to compute properties of data stream in low memory/space (and time)

$$\text{poly}(\log m, \log n)$$

Needed to index where we are in the stream

Needed to remember the current item

Sometimes, one can find algorithms that do not depend on the length of the stream m

In fact, streaming algorithms are sometimes also used for non-streaming data
E.g. To process data in massive datacenters, where data is stored on hard disks which are slow to read/write and one wants a low memory algorithm since we want to store the data relevant for the computation in the RAM

Some Examples

$\boxed{1}$ Addition (or Average)
Each item is a $O(\log n)$-bit number
Sum of m numbers is at most $m \cdot 2^{O(\log n)}$
so, we only need to store $O(\log m + \log n)$ bits

$\boxed{2}$ Max/Min
$O(\log n)$ bits

$\boxed{3}$ Median
Exact median requires $\Omega(n)$ space !

In-class Exercise

[1] Suppose the stream is $a_1, \ldots, a_n$ where each $a_i \in [n+1]$ and distinct

Find the missing value in $O(\log n)$ space

[2] Given a stream $\overbrace{a_1, \ldots, a_m}^{\text{distinct}}$, sample a uniformly random element from all the elements seen thus far, with only $O(\log n + \log m)$ space

Solution

[1] Missing value $= \sum_{i=1}^{n+1} i - \sum_{i=1}^{n} a_i$

[2] Let $s \leftarrow a_1$

When a_i arrives, with probability $\frac{1}{i}$, set $s \leftarrow a_i$

Why is s uniformly distributed?

$$\mathbb{P}[s = a_i] = \frac{1}{i}$$

$$\forall j < i: \ \mathbb{P}[s = a_j] = \left(1 - \frac{1}{i}\right) \cdot \frac{1}{i-1} = \frac{1}{i}$$

Bonus Exercise

Given a stream $a_1, \ldots, a_m$, sample a uniformly random set of s elements from all the elements seen thus far, with $O(s(\log n + \log m))$ space

Store $B = (a_1, \ldots, a_s)$ → B is a set of s elements

For $i > s$, with probability $\frac{s}{i}$ replace b_J with a_i

↑ J is chosen uniformly at random from $[s]$

Why does this give a uniformly random sample?

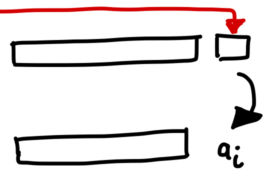
Consider any set b of s elements

We want to say that $\mathbb{P}[B=b] = \frac{1}{\binom{i}{s}}$

Suppose $a_i \notin b$, then
$$\mathbb{P}[B=b] = \frac{1}{\binom{i-1}{s}} \left(\frac{i-s}{i}\right) = \frac{s!\,(i-s-1)!}{(i-1)!} \cdot \frac{(i-s)}{i} = \frac{s!\,(i-s)!}{i!} = \frac{1}{\binom{i}{s}}$$

Suppose $a_i \in b$, then
$$\mathbb{P}[B=b] = \frac{(i-1)-(s-1)}{\binom{i-1}{s}} \cdot \frac{s}{i} \cdot \frac{1}{s} = \frac{\frac{(i-s)}{s}}{\binom{i-1}{s}} = \frac{1}{\binom{i}{s}}$$

$(i-1)-(s-1)$ choices for element that is replaced by a_i



Distinct Element Estimation

Given a stream $(a_1, a_2, \dots, a_m)$ where each $a_i \in \mathcal{U}$ with $|\mathcal{U}| = n$

Count the number of distinct elements in the stream, denoted F_0

Naive Algorithms $\boxed{1}$ Store an indicator vector of which elements of $\mathcal{U}$ we have seen

100110111000
← n bits →

$\boxed{2}$ Store a set of all the elements we recieve.
Space $O(m \log n)$ bits

Can we design a poly($\log m, \log n$) space algorithm ?

It turns out that both randomized and approximation are necessary to solve this problem

- Every deterministic algorithm requires $\Omega(n)$ bits, even for 1.1 approximation
- Every randomized algorithm that computes F_0 exactly requires $\Omega(m)$ bits

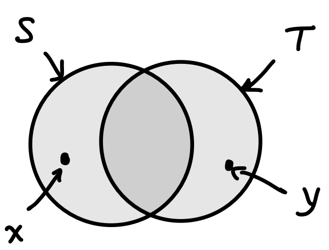
We will only prove a lower bound for exact deterministic algorithms here.

$\boxed{\text{Lemma}}$ Exactly counting number of distinct elements requires $\Omega(m)$ space (assuming $n \geq 2m$).

Proof Suppose the first $m-1$ elements are distinct and algorithm uses s bits of memory
There are $\binom{|\mathcal{U}|}{m-1}$ choices of inputs for the first $(m-1)$ elements

And 2^s choices for memory configurations

If $\binom{|\mathcal{U}|}{m-1} > 2^s$, then there must be two sets that lead to the same memory configuration. Let the two sets be S & T where



The algorithm must err in one of the two input streams since the memory configuration is the same and

- $S \cup \{x\}$ $\longrightarrow$ # distinct elements $= m-1$ $\longleftarrow$ $T \cup \{y\}$
- $S \cup \{y\}$ $\longrightarrow$ # distinct elements $= m$ $\longleftarrow$ $T \cup \{x\}$

Thus, $2^s \geq \binom{2m}{m-1} \implies s = \Omega(m)$ $\square$

Approximately Counting # Distinct Elements with Randomized Algorithms

$\boxed{\text{Goal}}$ Given a stream $(a_1, \ldots a_m)$ design a randomized algorithm that outputs a number D s.t.

$$\mathbb{P}\left[\, D \in [(1-\varepsilon)F_0, (1+\varepsilon)F_0] \,\right] \geq 1-\delta$$

[Kane, Nelson, Woodruff '10] gave an algorithm with space $O\left(\left(\frac{1}{\varepsilon^2} + \log n\right) \cdot \log \frac{1}{\delta}\right)$

This algorithm is best possible in terms of space complexity

Beyond the scope of this course.

Today, we will see a simple algorithm with space complexity $O\left(\frac{\log n}{\varepsilon^2} \cdot \log\left(\frac{m}{\delta}\right)\right)$

The algorithm is due to [Chakraborty-Vinodchandran-Meel '23]

The basic idea behind the algorithm is the following :

- Suppose we randomly sample a set X where each distinct element in the stream is included with probability p independently.

$1 \; 2 \; 3 \; 4 \; 2 \; 3 \; 5 \; 4 \; 7$
$\downarrow \; \downarrow \; \downarrow \; \downarrow \qquad \downarrow \quad \downarrow$
each included in X independently with probability p

Then, $\mathbb{E}[|X|] = p \cdot F_0 \iff \mathbb{E}\left[\frac{|X|}{p}\right] = F_0$

Furthermore, by Chernoff bounds

$$\mathbb{P}\Big[\underbrace{|X - pF_0| \geq \varepsilon p F_0}_{= \left|\frac{|X|}{p} - F_0\right| \geq \varepsilon F_0}\Big] \leq e^{-\varepsilon^2 \cdot \mathbb{E}[|X|]} = e^{-\varepsilon^2 p F_0}$$

Thus, we can just randomly sample a set X as above, divide its size by p and hope to get the value of F_0, as long as p is not too small

$\hookrightarrow$ Want $p = \frac{100}{\varepsilon^2 F_0} \log\left(\frac{m}{\delta}\right)$

so, that $e^{-\varepsilon^2 p F_0} \leq \frac{\delta}{4m}$

& $\mathbb{E}[\text{size}] = \frac{100}{\varepsilon^2} \log$

There are only two problems here :

<u>Sampling</u> How might one sample such a set ?

<u>Rate of Sampling</u> The Chernoff bound calculation suggested that we don't want p to be too small.

But we don't want p to be too large either since we want X to have small size, so we can store it with small space.

Ideally, we would want $p \approx \frac{1}{F_0}$, so that $\mathbb{E}|X| \approx 1$, but we don't know F_0 !!

Let's see how to resolve these problems one by one :

Sampling Let the current set be X and the next element be a_i

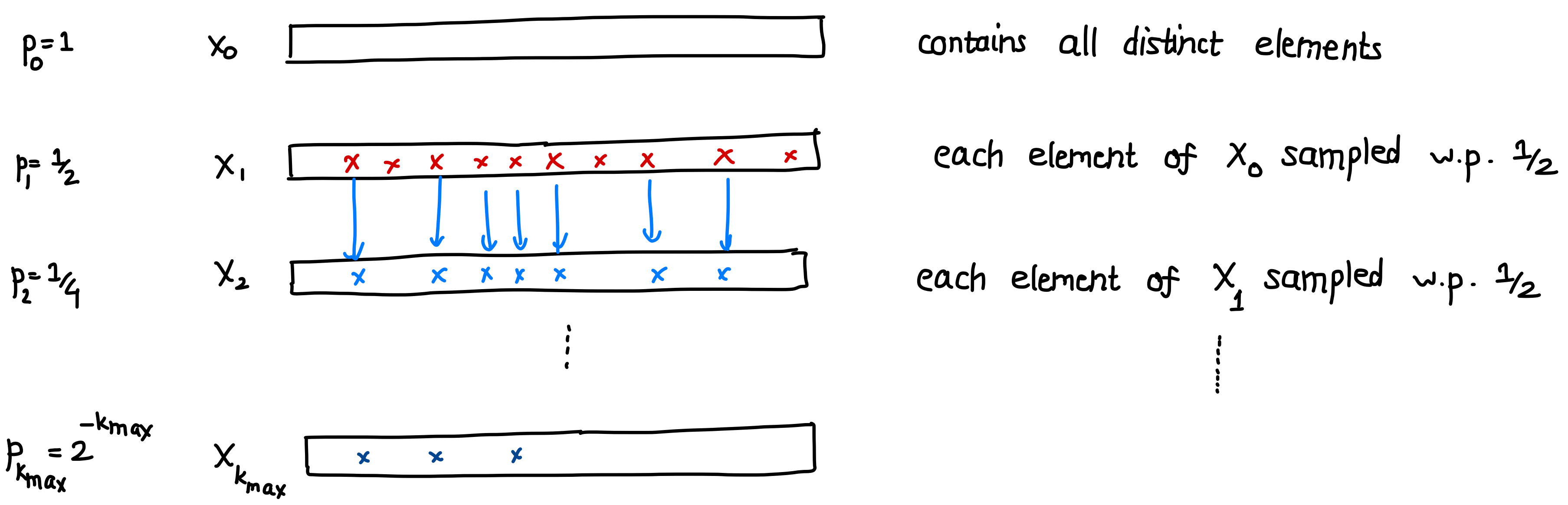
> Remove a_i from X if it occurs
> Then, add a_i to X with probability p

$\boxed{\text{Claim}}$ Let the distinct elements seen in the stream $(a_1, \ldots, a_i)$ be Y. Then, X is a random subset obtained by sampling each element of Y with probability p independently.

Proof Exercise

Rate of Sampling The key idea is to try all rates $p_k = 2^{-k}$ for different values of k

Maintain X_k sampled at rate 2^{-k} from distinct elements



As long as the set $X_{k_{max}}$ has not too small a size, we can use any of these sets to estimate F_0, by using the associated rate

But storing each set may still require a lot of space !!

We only need one such set however with the associated rate !

In particular, we keep a threshold of our bucket of size $\frac{100}{\varepsilon^2} \log\left(\frac{m}{\delta}\right)$

If the bucket exceeds this size we throw away that bucket & move to the next one & keep track of the value of p

Overall, our algorithm is the following

Estimate F_0 ($a_1, \ldots, a_m$)

$p \leftarrow 1$, $X \leftarrow \emptyset$

For $i \leftarrow 1$ to m

$X \leftarrow X \setminus \{a_i\}$

with probability p, $X \leftarrow X \cup \{a_i\}$] sample from distinct elements at rate p

if $|X| = \frac{100}{\varepsilon^2} \log\left(\frac{m}{\delta}\right)$, then

throw away each element of X with probability $\frac{1}{2}$

$p \leftarrow p/2$] subsample half the elements & decrease the rate

Output $\frac{|X|}{p}$

$\boxed{\text{Lemma}}$ The probability that $p \leq \dfrac{\frac{100}{\varepsilon^2}\log\left(\frac{m}{\delta}\right)}{4F_0}$ at any point during the run of the algorithm is at most δ.

$\Longrightarrow$ This implies that the size of the set at the end is large enough, so that Chernoff bounds imply that we output a $(1 \pm \varepsilon)$ approximation of F_0

And also space complexity is $O\left(\log n \cdot \frac{100}{\varepsilon^2} \log\left(\frac{m}{\delta}\right)\right)$

Proof of Lemma

Suppose the probability decreases from $2^{-\ell}$ to $2^{-\ell-1}$ where $2^{-\ell-1} \leq \dfrac{\frac{100}{\varepsilon^2}\log\left(\frac{m}{\delta}\right)}{4F_0}$

This can only happen when the subsampled set X at this rate has reached maximum bucket size.

However, $\mathbb{E}|X| \leq \frac{100}{\varepsilon^2}\log\left(\frac{m}{\delta}\right) \cdot \frac{1}{4\cancel{F_0}} \cdot \cancel{F_0} = \frac{25}{\varepsilon^2}\log\left(\frac{m}{\delta}\right)$

Thus, by Chernoff bounds

$$\mathbb{P}\left[\,|X| \geq \frac{100}{\varepsilon^2}\log\left(\frac{m}{\delta}\right)\right] \leq e^{-c \cdot \frac{1}{\varepsilon^2}\log\left(\frac{m}{\delta}\right)} \leq \frac{\delta}{m}$$

By union bound over all m iterations, the probability that p decreases below the above threshold is at most δ