COMS E6998-9: Algorithms for Massive Data (Spring'19) Jan 29, 2019

Lecture 3: Impossibility, Frequency moments

Instructor: *Alex Andoni* Scribes: *Daniel Mitropolsky*

1 Administrivia

- Problem Set 1 due on Febuary 14
- See website for policy on concise proofs and collaboration (with up to 2 collaborators, independent write-ups!)

Office hours:

- Prof. Andoni: Mondays 3:30-5:30
- Peilin: Tuesdays 10:00-12:00
- Marshall: Wednesdays 3:00-5:00

2 Review

Distinct element (DE) counting
• Given a stream $x_1, \dots, x_n \in [n]$ count the number of distinct elements.
• **Theorem**: there exists an algorithm that outputs with 90% probability a $(1+\epsilon)$-approximation with space $O(\frac{1}{\epsilon^2})$ words.
• Also need space for the storing hashing functions: if we discretize them, we need $\log n$ bits / word (with more work we can discretize better and only use $\log \log n$ bits, giving rise to the popular "hyperloglog" algorithm)
• For the hash functions we don't need full randomness. Pairwise independent functions are enough.

3 Impossibility results

Today we will argue that we need the relaxations of approximation and randomness to get $O(\log n)$ bits, instead of n bits for an exact algorithm. We will show the relaxations are necessary (some today, others will be in homework).

Theorem 1. *Any algorithm for DE that is deterministic and exact must use n bits.*

This is a lower bound, not an analysis of an algorithm. It is similar to showing the lower bound for the number of comparisons in sorting to be $O(n \log n)$.

Proof. Proof by contradiction. Fix an algorithm A that manages to compute the number of distinct elements, and is deterministic and exact.

Consider a binary vector $y \in \{0,1\}^n$. Let S_y be the stream such that $x = i$ is in the stream iff $y_i = 1$.

Example 2. *If* $y = (0,1,1)$ *then* $S_y = (2,3)$.

Now run A on S_y. A is a finite algorithm receiving inputs one-by-one, jumping between its states. If A is a deterministic finite automaton, then at any time its state is equivalent to what is stored in its memory.

Let $\sigma_{A,y}$ be the memory contents of A after running on S_y. The length $|\sigma_{A,y}| \leq s$, which we define to be the size of the memory of A.

Define a function $f : \{0,1\}^n \to \{0,1\}^s$ as $f(y) = \sigma_{A,y}$.

Claim 3. f *is injective.*

Proof. Given $\sigma_{A,y}$, we can recover the vector y from it, proving f is injective.

Fix A after seeing S_y. It is in state $\sigma_{A,y}$. Feed A another stream element $x = i$. The estimate of the number of distinct elements increases $\iff y_i = 0$.

Hence we can recover y_i from $\sigma_{A,y}$, because after S_y the only state that is stored is the memory contents of A.

This implies we can recover the entire vector $y \in \{0,1\}^n$. Since $\sigma_{A,y} \implies y$, we can invert f so it is injective. □

This claim implies the domain of $f : \{0,1\} \to \{0,1\}^s$ is a lower bound for the range.

$\implies 2^n \leq 2^s$

$\implies n \leq s$

This concludes the proof of Theorem 1. □

The subtlety of this proof was in the construction of a function f that takes y, feeds the stream to the algorithm, and outputs the state σ of the algorithm after the stream. Then from this state σ we recover y, proving f is injective.

Theorem 4. *If A is deterministic and a* 1.1*-approximation, it still needs* $\Omega(n)$ *space.*

Proof. We will again try to define a function and prove it is injective.

Let the set $T \subset \{0,1\}^n$ represent a code with the following conditions:

- $\forall x \neq y \in T$, $|y \backslash x| \geq n/6$ (where $\backslash$ is set subtraction of the vectors viewed as sets).
- $\forall x \in T$, $\|x\|_1 = n/2$ (balanced)
- $|T| = 2^{\Omega(n)}$ (set T is very large).

A theorem of Shannon guarantees that a random set T if size $2^{\Omega(n)}$ satisfies these conditions; we do not construct this set but we assume it exists.

Build $f : T \to \{0,1\}^s$ where again $s =$ the space of A. Define $f(y) = \sigma_{A,y}$ be the space contents of A after running on S_y (for $y \in T$). Exactly as we did before, this yields an encoding of the stream y to binary vectors of length s.

We will prove in the next claim that f is injective. Then, $|T| \leq 2^s \implies s \geq \Omega(\log T) = \Omega(n)$.

Claim 5. *f is injective.*

Proof. We can think of f as an encoding taking a vector in T to a string of length s. We want to decode $y \in T$ from the memory space $\sigma_{A,y}$.

Let $d =$ the number of distinct elements estimated by A with memory $\sigma_{A,y}$.

For each $x \in T$ (representing a guess for the vector y), now stream through S_x.

Let d' be the estimate of distinct elements after also streaming S_x.

We claim that $d' < \left(\frac{n}{2} + \frac{n}{6}\right)/1.1$, then $y = x$. There are two cases to check:

1. If $x = y$, then
   - # distinct elements in S_y, S_x is $\frac{n}{2}$
   - $\implies d' < \frac{n}{2} \cdot 1.1 < \left(\frac{n}{2} + \frac{n}{6}\right)/1.1$
2. if $x \neq y$,. then
   - # distinct e;ements in S_y, S_x is $\geq \frac{n}{2} + \frac{n}{6}$
   - $\implies d' \geq \left(\frac{n}{2} + \frac{n}{6}\right)/1.1$

Hence, if $d' < \left(\frac{n}{2} + \frac{n}{6}\right)/1.1$ we can output that $y = x$ and halt our search for the value of y. This proves that f is injective. ☐

With this claim ends the proof of the theorem. ☐

Why would this argument fail for a randomized algorithm? We assumed d' is correct up to approximation. If A is random, that is now always true and we may return $y = x$ that is wrong because the procedure failed.

Observe that the proof would hold for any fixed $(1 + \epsilon)$-approximation.

4 Frequency moments

Let $x_1 \ldots, x_n \in [n]$ be a stream. Define $f \in \mathbb{N}^n$ where $f_i = \#$ times i appears in the stream.

Definition 6. *For $p \in \mathbb{N} \cup \{\infty\}$, the p-th moment F_p is*

$$F_p = \sum_{i=1}^{n} f_i^p = \|f\|_p^p$$

Example 7. *$F_1 =$ sum of frequencies $= m$ (length of the stream)*

Example 8. *$F_0 = \#$ non-zero items, which is the # distinct elements*

In general, frequency moments characterize the stream, giving statistics about the whole stream.

We define F_∞ as $\|f\|_\infty = \lim_{p \to \infty} \left(\sum f_i^p\right)^{1/p}$. We will talk about F_∞ later in the course.

The second moment F_2 is related of the variance of the stream, a proxy for how imbalanced the frequencies are. It is useful and we will see that it is related to dimensionality reduction.

4.1 "Tug-of-wars" algorithm for F_2 [Alon-Matias-Szegedy '96]

The idea is to use random variablies (or hash function) $r : [n] \to \{\pm 1\}$ called *Rademacher random variables* from functional analysis.

Algorithm:

- Mathematical description: store $z = \sum_{i=1}^{n} r_i f_i$.
- Operational description: on update $x = i$, set $z = z + r_i$.
- Estimate: return z^2

Observation 9. $\mathbb{E}_r[z] = \mathbb{E}_r\big[\sum r_i f_i\big] = \sum_i f_i \mathbb{E}_r[r_i] = 0$, *so z is a $0-$centered random variable.*

This is some intuition for why we use z^2 as the estimator and not z. Now we analyze,

$$\begin{aligned}
\mathbb{E}_r[z^2] &= \mathbb{E}_r\big[\big(\sum_i r_i f_i\big)^2\big] \\
&= \mathbb{E}_r\big[\sum_{i,j} r_i r_j f_i f_j\big] \\
&= \sum_{i,j} f_i f_j \mathbb{E}[r_i][r_j] \\
&= \sum_i f_i f_i \\
&= F_2
\end{aligned}$$

Where we used that $\mathbb{E}[r_i r_j] = 1$ iff $i = j$ and 0 otherwise because r_i and r_j are independent (so $\mathbb{E}[r_i r_j] = \mathbb{E}[r_i]\mathbb{E}[r_j] = 0$).

This method won't for for moments beyond 2; even for $p = 2.1$ we need space that is poly(n). L_2, Euclidean space, has a powerful structure no other p has.

Now we want to compute the Var.

$$\begin{aligned}
\mathrm{Var}\,[z^2] &\le \mathbb{E}_r[z^4] \\
&= \mathbb{E}_r\big[(r_1 f_1 + r_2 f_2 + \cdots + r_n f_n) \times (r_1 f_1 + r_2 f_2 + \cdots + r_n f_n) \times (\cdots) \times (\cdots)\big]
\end{aligned}$$

Consider expanding the product; you get n^4 monomials of four terms. Those that don't vanish are those where the terms are all of the same sort, or 2 of 2 sorts. Note that $\mathbb{E}[r_i^4] = 1$ and $\mathbb{E}[r_i^2 r_j^2] = 1$. Hence we have,

$$\begin{aligned}
\mathrm{Var}\,[z^2] &= \sum_i f_i^4 \mathbb{E}[r_i^4] + 3\sum_{i \neq j} f_i^2 f_j^2 \mathbb{E}[r_i^2 r_j^2] \\
&\le \sum_i f_i^4 + 3\big(\sum_i f_i^2\big)^2 \\
&\le 4\big(\sum_i f_i^2\big)^2 \\
&= 4F_2^2
\end{aligned}$$

By Chebyshev bound, we have that $z^2 \in F_2 \pm \sqrt{10F_2^2}$ with 90% probability (i.e., $z^2 \in F_2 \pm \sqrt{10}F_2$). Keep $k = \frac{1}{\epsilon^2}$ ToW counters $z_1, \ldots, z_k$. Our estimator will now be

$$E = \frac{1}{k}\sum_{i=1}^{k} z_i^2$$

Using the same proposition we proved in the first lecture,

Claim 10. $\mathbb{E}[E] = F_2$ *and* $\mathrm{Var}[E] \leq \frac{4}{k}F_2^2$.

By Chebyhsev, we have $E \in F_2 \pm \sqrt{10/k}F_2$ with 90% probability. The multiplicative bound is ϵ when $k = 40/\epsilon^2$.

We are able to get a $(1+\epsilon)$-approximation using $O(1/\epsilon^2)$ counters. Each is a sum of the $r_i f_i$, so $\log n$ bits is enough for each!

What about r? It would seem to take up $O(n)$ space, defeating the purpose. From the analysis, it is clear that a 4-wise independent hashing function is enough.