CS 395T: Sublinear Algorithms Fall 2016

Lecture 3 — September 1, 2016

Prof. Eric Price *Scribe: William Hoza*

1 Overview

We continue discussing the *distinct elements* problem. Recall that in this problem, you receive as input a stream $S = (x_1, x_2, \dots)$, with each $x_i \in [n]$, and you are supposed to compute $k = \#\{v \in [n] : v \text{ appears in } S\}$. Topics for today:

- Review of the algorithm from last lecture.
- The LogLog algorithm, which is more space efficient.
- Generalizing the algorithm from last lecture to work in the *turnstile model*.
- Space complexity *lower bounds* via a connection with *one-way communication complexity*.

2 Review of last lecture

Recall the algorithm from last lecture:

1. For $t = 1, 2, 4, 8, \dots, n$, in parallel:
   (a) For $r = 1$ to $R \leq O(\log \frac{\log n}{\delta})$, in parallel:
      i. Pick a random hash function $h_{t,r} : [n] \to [10t]$ from pairwise independent family $\mathcal{H}$.
      ii. Record $Y_{t,r} = \begin{cases} 1 & \text{if some } v \in S \text{ satisfies } h_{t,r}(v) = 0 \\ 0 & \text{otherwise.} \end{cases}$
   (b) Compute the empirical average $\hat{Y}_t = \frac{1}{R} \sum_{r=1}^{R} Y_{t,r}$.
   (c) Set $z_t = \begin{cases} 1 & \text{if } \hat{Y}_t > 0.14 \\ 0 & \text{otherwise.} \end{cases}$
   (d) Output $2^{\sum_t z_t}$.

Analysis We showed last time that if $k < t$, then $\mathbb{P}[z_t = 0] \geq 1 - \frac{\delta}{\log n}$, while if $k > 2t$, then $\mathbb{P}[z_t = 1] \geq 1 - \frac{\delta}{\log n}$. Therefore, by the union bound, the output of the algorithm is within a factor of 2 of k with probability at least $1 - \delta$. The space used by this algorithm is

$$O((\log^2 n) \cdot (\log \log n + \log(1/\delta))).$$

3 The LogLog algorithm

Suppose we pick a function $h : [n] \to [0, 1]$ at random, and then compute $Y = \min_{v \in S} h(v)$. How does Y relate to k? Intuitively, one would expect that Y would be about $1/k$, and indeed one can show with calculus that $\mathbb{E}[Y] = \frac{1}{k+1}$. This suggests that to compute k, we should output $1/Y - 1$. Unfortunately, Y has too much variance for this to be a good idea. Naturally, we can deal with the variance by taking several independent trials. This is the idea behind the LogLog algorithm. Fix $\epsilon > 0$; the algorithm will output a $(1 + \epsilon)$-approximation, except for a $1/4$ probability of failure. Here is the algorithm:

1. For $i = 1$ to $m \leq O(1/\epsilon^2)$, in parallel:
   (a) Pick a random function $h_i : [n] \to [0, 1]$.
   (b) Record $Y_i = \min_{v \in S} h(v)$.
2. Compute $\hat{Y} = \frac{1}{m} \sum_{i=1}^{m} Y_i$.
3. Output $1/\hat{Y} - 1$.

Proof of correctness For each i, the probability that Y_i exceeds its expectation by a factor of c decays *exponentially* with c:

$$\mathbb{P}\left[Y_i > \frac{c}{k}\right] = \left(1 - \frac{c}{k}\right)^k \leq e^{-c}.$$

Therefore,

$$\mathbb{E}[Y_i^2] = \int_0^\infty \mathbb{P}[Y_i^2 \geq t]\, dt \leq \frac{1}{k^2} \cdot \int_0^\infty e^{-\sqrt{t}}\, dt = \frac{2}{k^2}.$$

Since $\mathrm{Var}(Y_i) \leq \mathbb{E}[Y_i^2]$, it follows that $\mathrm{Var}(Y_i) \leq \frac{2}{k^2}$. It follows that

$$\mathrm{Var}(\hat{Y}) = \mathrm{Var}\left(\frac{1}{m}\sum_{i=1}^{m} Y_i\right) = \frac{1}{m^2}\mathrm{Var}\left(\sum_{i=1}^{m} Y_i\right) = \frac{1}{m^2}\sum_{i=1}^{m}\mathrm{Var}(Y_i) \leq \frac{2}{k^2 m}.$$

Obviously $\mathbb{E}[\hat{Y}] = \mathbb{E}[Y_i] = \frac{1}{k+1}$, so by Chebychev's inequality, for any $t > 0$,

$$\mathbb{P}\left[\left|\hat{Y} - \frac{1}{k+1}\right| > t\right] \leq \frac{2}{k^2 m t^2}.$$

Plugging in $t = \epsilon/k$ gives

$$\mathbb{P}\left[\left|\hat{Y} - \frac{1}{k+1}\right| > \frac{\epsilon}{k}\right] \leq \frac{2}{m\epsilon^2}.$$

If $m = 8/\epsilon^2$, this is $\leq 1/4$. All that remains is to show that our choice of t was good enough to guarantee that the algorithm's output is approximately correct. In the case that $\hat{Y}$ is within t of its expectation, we can *upper* bound the algorithm's output by

$$1/\hat{Y} - 1 \leq \frac{1}{\frac{1}{k+1} - \frac{\epsilon}{k}} - 1 \leq \frac{k + 2\epsilon}{1 - 2\epsilon} \leq (1 + 3\epsilon)k + 3\epsilon$$

for small enough ϵ, which is at most $(1 + 6\epsilon)k$. A similar analysis will *lower* bound the algorithm's output. By replacing ϵ with $\epsilon/6$ everywhere, we see that $m \leq O(1/\epsilon^2)$ trials suffice to achieve a $1 + \epsilon$ approximation with $1/4$ probability of failure.

Space efficiency How much space do we need to use for storing the h_i functions? In the random oracle model, none. The random oracle model can be avoided by using appropriate "min-wise independent" hash functions, but we'll skip that. How much space do we need to record each Y_i value? Of course, we can't actually store Y_i to infinite precision; we will need to choose our method for representing an *approximation* of Y_i in a space-efficient manner.

We need a representation which allows us to compare two numbers and which guarantees good multiplicative accuracy. One simple strategy would be to store the first $O(\log n)$ bits of the binary expansion of Y_i. This would give $\pm\frac{1}{n^c}$ *additive* error. The multiplicative error would be at most $1 \pm \frac{1}{n^c}$ assuming $Y_i \geq \frac{1}{n^c}$, which occurs with high probability.

A smarter strategy exploits the fact that as k increases, the Y_i values will be decreasing. We just store the order of magnitude, i.e. the number of leading zeroes. This requires only $O(\log \log n)$ bits, as long as we cap the number of zeroes at $O(\log n)$. One can show that even with the error introduced by rounding, the algorithm has a low probability of failure.

HyperLogLog is a practical variant of the LogLog algorithm. For example, with $n = 10^9$, using 1.5 kB of memory, the HyperLogLog algorithm achieves $\epsilon = 2\%$.

4 The turnstile model

We now generalize to the *turnstile model*. In this model, the algorithm tracks a vector $x \in \mathbb{R}^n$ (think of it as a histogram.) Initially, $x = 0$. The algorithm is given *updates* of the form (i, α), meaning that x_i is replaced with $x_i + \alpha$. At the end, the algorithm should output $f(x)$ for some function f. For example, in the distinct elements problem, we are trying to compute

$$f(x) = \|x\|_0 = \#\{i : x_i \neq 0\}.$$

So far, we've only considered "insertion streams," where $\alpha = 1$ every time. It is also natural to consider deletions ($\alpha = -1$), and in fact the model makes sense even if we allow arbitrary $\alpha \in \mathbb{R}$. In the *strict turnstile model*, we assume that $x_i \geq 0$ for all i at all times.

4.1 Distinct elements in the turnstile model

There's no clear way to generalize the LogLog algorithm. On the other hand, the algorithm from last lecture readily admits generalization. Rather than defining $Y_{t,r}$ to indicate whether *some* $v \in S$ satisfies $h_{t,r}(v) = 0$, we can define $Y_{t,r}$ to be the *number* of $v \in S$ satisfying $h_{t,r}(v) = 0$. Explicitly, when we are given (v, α), if $h(v) = 0$, we set $Y_{t,r} \leftarrow Y_{t,r} + \alpha$, thereby ensuring that

$$Y_{t,r} = \sum_{\substack{v \in [n] \\ h(v)=0}} x_v.$$

In the strict turnstile model, this works well, because at the end of the updates, we can just check whether $Y_{t,r} > 0$. In the non-strict turnstile model, this idea doesn't make so much sense, because an insertion and a deletion might cancel each other out.

5 Lower bounds

The algorithms we've seen for the distinct elements merely give *approximately* the right answer ($\epsilon > 0$), and they have had some small *failure probability* ($\delta > 0$.) Unfortunately, both of these defects are unavoidable! In class, we will prove that $\delta > 0$ is necessary; on the homework, we will prove that $\epsilon > 0$ is necessary. These lower bounds hold even against the insertion-only model.

Theorem 1. *Fix any deterministic $(1 + 0.1)$-approximation algorithm for the distinct elements problem in the insertion-only streaming model. For any $k, n \in \mathbb{N}$ with $k \leq n$, there is a stream over the universe $[n]$ with $\leq k$ distinct elements on which the algorithm uses $\Omega(k)$ space.*

How do we prove a theorem like this? Typically, lower bounds for streaming algorithms come from *one-way communication complexity.*

5.1 One-way communication complexity

The setup here is that Alice has x, Bob has y, Alice sends m, and Bob outputs $f(x, y)$. For various f, there are known lower bounds on $|m|$ for any protocol in which Bob gets the right answer with (say) 3/4 probability.

As a trivial example, computing the function $f : X \times Y \to X$ defined by $f(x, y) = x$ requires $\Omega(\log |X|)$ bits of communication. For a less trivial example, consider the *indexing* problem, where Alice holds $x \in \{0, 1\}^n$, Bob holds $i \in [n]$, and the goal is to compute $f(x, i) = x_i$.

Theorem 2. *In any protocol for the indexing problem with failure probability at most 1/8, Alice sends $\Omega(n)$ bits to Bob.*

We'll need an elementary fact from coding theory:

Lemma 3. *For every n, there exists $\mathcal{X} \subseteq \{0, 1\}^n$ with minimum Hamming distance more than $n/3$ with $|\mathcal{X}| \geq \exp(\Omega(n))$.*

Proof of Lemma 3. Start with $\mathcal{X} = \emptyset$. Repeatedly add a string $x \in \{0, 1\}^n$ with $\text{dist}(x, \mathcal{X}) > n/3$ as long as such a string exists. By the Chernoff bound, for any $y \in \{0, 1\}^n$, if we pick $x \in \{0, 1\}^n$ uniformly at random, then

$$\mathbb{P}[\text{dist}(x, y) \leq n/3] \leq \exp(-\Omega(n)).$$

So by the union bound, at any stage in this process,

$$\mathbb{P}[\text{dist}(x, \mathcal{X}) \leq n/3] \leq |\mathcal{X}| \cdot \exp(-\Omega(n)).$$

As long as this probability is less than 1, the process will continue. Therefore, we can achieve $|\mathcal{X}| \geq \exp(\Omega(n))$. □

Proof of Theorem 2. By reduction from the trivial example. Let $\mathcal{X} \subseteq \{0, 1\}^n$ be as in Lemma 3. Alice gets $x \in \mathcal{X}$ and sends whatever she would send in the indexing protocol. Bob simulates his behavior in the indexing protocol on every $i \in [n]$, obtaining a string $y \in \{0, 1\}^n$. Each position of y has only a 1/8 chance of disagreeing with the corresponding position of x, so by the Chernoff bound, with high probability, $\text{dist}(x, y) < n/6$. In this case, Bob can recover x just by finding the nearest codeword in $\mathcal{X}$ to y. Hence, Alice must have sent $\Omega(\log |\mathcal{X}|) = \Omega(n)$ bits. □

Remark In the *augmented indexing* problem, Bob gets $x_1, x_2, \ldots, x_{i-1}$ in addition to i. It turns out that the augmented indexing problem still requires $\Omega(n)$ bits of communication. Observe that if Bob additionally got $x_{i+1}, \ldots, x_n$, Alice could just send the parity of x (one bit).

5.2 Connection with streaming algorithms

Proof of Theorem 1. By reduction from the trivial communication complexity example. For a warm-up, consider a distinct elements algorithm with $\epsilon = \delta = 0$. Alice gets $x \in \{0,1\}^k$. She uses this to construct a subset $S_x = \{(i, x_i) : 1 \leq i \leq k\}$, which we can think of as a subset of the universe $[n]$ for any $n \geq 2k$. She feeds S_x to the streaming algorithm and sends the sketch (i.e. the state of the algorithm) to Bob.

To check if $x = y$, Bob feeds S_y to the algorithm. If $x = y$, the number of distinct elements won't change. But if $x \neq y$, the number of distinct elements will increase by at least 1. By trying every y, Bob can learn x. Hence, Alice must have sent $\Omega(k)$ bits.

Now, to prove the theorem, let's deal with $\epsilon = 0.1$, $\delta = 0$. Let $\mathcal{X} \subseteq \{0,1\}^k$ be as in Lemma 3. Alice gets $x \in \mathcal{X}$, and they proceed as before, except that Bob only checks $y \in \mathcal{X}$. If $x = y$, again, the number of distinct elements remains k, but if $x \neq y$, then the number of distinct elements will go up to at least $(4/3)k$. These two cases can be distinguished by our approximation algorithm, so again, Bob can learn x. Hence, Alice must have sent $\Omega(\log |\mathcal{X}|) = \Omega(k)$ bits. □