Advanced Algorithms November 16, 2024

Lecture 18: Distinct Elements Continued

Notes by Ola Svensson[1]

1 Estimating the Number of Distinct Elements

DISTINCT-ELEMENTS problem Our goal is to output an approximation to then number $d(\sigma) = |\{j : f_j > 0\}|$ of distinct elements that appear in the stream σ.

It is provably impossible to solve this problem in sublinear space if one is restricted to either deterministic algorithms or exact algorithms. Thus we shall seek a randomized approximation algorithms. More specifically, we give a guarantee of the following type

$$\Pr[d(\sigma)/3 \leq A(\sigma) \leq 3d(\sigma)] \geq 1 - \delta,$$

i.e., with probability $1 - \delta$ we have a 3-approximate solution. (With more work the 3 can be improved to $1 + \varepsilon$ for any $\varepsilon > 0$.) The amount of space we use will be $O(\log(1/\delta) \log n)$.

Recall that we use the $zeros(p)$ function: for an integer $p > 0$ let $zeros(p)$ denote the number of zeros that the binary representation of p ends with. Formally,

$$zeros(p) = \max\{i : 2^i \text{ divides } p\}.$$

Examples are $zeros(2) = 1, zeros(3) = 0, zeros(4) = 2, zeros(6) = 1, zeros(7) = 0$.

The algorithm is

Initialization: Choose a random hash function $h : [n] \to [n]$ from a pairwise independent family[2]. Let $z = 0$.

Process j: If $zeros(h(j)) > z$ then $z = zeros(h(j))$.

Output: $2^{z+1/2}$.

We only use $O(\log n)$ space. The intuition for correctness is very simple:

- The probability that a random number x has $zeros(x) \geq \log d$ is $1/d$.
- So if we have d distinct numbers we would expect $zeros(h(j)) \geq \log d$ for some element j.

Let's now analyze the quality of the output.

1.1 Analysis of Algorithm

- For each $j \in [n]$ and each integer $r \geq 0$, let $X_{r,j}$ be an indicator random variable for the event "$zeros(h(j)) \geq r$," and let $Y_r = \sum_{j:f_j>0} X_{r,j}$.
- Let t denote the value of z when algorithm terminates. By definition,

$$Y_r > 0 \iff t \geq r. \tag{1}$$

[1] **Disclaimer:** These notes were written as notes for the lecturer. They have not been peer-reviewed and may contain inconsistent notation, typos, and omit citations of relevant works.

[2] To ease calculations we assume that n is a power of two and so we hash a value p to a uniformly at random binary string of length $\log_2(n)$.

- It will be useful to restate this fact as follows:

$$Y_r = 0 \iff t \leq r - 1. \tag{2}$$

- Since $h(j)$ is uniformly distributed over $(\log n)$-bit strings, we have

$$\mathbb{E}[X_{r,j}] = \Pr[zeros(h(j)) \geq r] = \Pr[2^r \text{ divides } h(j)] = \frac{1}{2^r}.$$

We now estimate the expectation and variance of Y_r. We have

$$\mathbb{E}[Y_r] = \sum_{j:f_j>0} \mathbb{E}[X_{r,j}] = \frac{d}{2^r}$$

and

$$\begin{aligned}
\mathrm{Var}[Y_r] &= \mathbb{E}[Y_r^2] - \mathbb{E}[Y_r]^2 \\
&= \mathbb{E}[\sum_{j,j':f_j,f_{j'}>0} X_{r,j}X_{r,j'}] - \sum_{j,j':f_j,f_{j'}>0} \mathbb{E}[X_{r,j}]\,\mathbb{E}[X_{r,j'}] \\
&= \sum_{j:f_j>0} \left(\mathbb{E}[X_{r,j}^2] - \mathbb{E}[X_{r,j}]^2\right) \\
&\leq \sum_{j:f_j>0} \mathbb{E}[X_{r,j}^2] = \sum_{j:f_j>0} \mathbb{E}[X_{r,j}] = \frac{d}{2^r}
\end{aligned}$$

Here, we used the pairwise independence from the hash-functions in the third equality.

Then by using Markov's inequality, we have

$$\Pr[Y_r > 0] = \Pr[Y_r \geq 1] \leq \frac{\mathbb{E}[Y_r]}{1} = \frac{d}{2^r} \tag{3}$$

(Recall that Markov's inequality says that for a non-negative random variable, we have $\Pr[Z \geq k] \leq \frac{\mathbb{E}[Z]}{k}$.)

Also by using Chebyshev's inequality, we have

$$\Pr[Y_r = 0] \leq \Pr\left[|Y_r - \mathbb{E}[Y_r]| \geq \frac{d}{2^r}\right] \leq \frac{\mathrm{Var}[Y_r]}{(d/2^r)^2} \leq \frac{2^r}{d}. \tag{4}$$

(Recall that Chebyshev's inequality says that for a random variable Z, $\Pr[|Z - \mathbb{E}[Z]| \geq k] \leq \frac{\mathrm{Var}[Z]}{k^2}$.)

- Let $\hat{d}$ be the estimate of d that the algorithm outputs. Then $\hat{d} = 2^{t+1/2}$.
- Let a be the smallest integer such that $2^{a+1/2} \geq 3d$. Using Equations (1) and (3),

$$\Pr[\hat{d} \geq 3d] = \Pr[t \geq a] = \Pr[Y_a > 0] \leq \frac{d}{2^a} \leq \frac{\sqrt{2}}{3}.$$

- Similarly, let b the largest integer such that $2^{b+1/2} \leq d/3$. Using Equations (2) and (4),

$$\Pr[\hat{d} \leq d/3] = \Pr[t \leq b] = \Pr[Y_{b+1} = 0] \leq \frac{2^{b+1}}{d} \leq \frac{\sqrt{2}}{3}.$$

These guarantees are pretty weak in two ways:

- First the estimate $\hat{d}$ is not arbitrarily close to d (can be fixed but not today).
- Secondly, the failure bounds (on each side) are $\frac{\sqrt{2}}{3} \approx 47\%$ which is high. How can we fix this problem? Clearly we could aim for a worse than 3-approximation and therefore obtain better failure probabilities.
- But a better idea, that does not further degrade the quality of the estimate $\hat{d}$, is to use a standard "median trick" which is really useful to know and use.

1.2 The Median Trick

- Imagine running k copies of this algorithm in parallel, using mutually independent random hash functions, outputting the median of the k answers.

If this median exceeds $3d$ then $k/2$ of the individual answers must exceed $3d$, whereas we only expect $\leq k\frac{\sqrt{2}}{3}$ of them to exceed $3d$. By a standard *Chernoff bound*, this event has a probability $\leq 2^{-\Omega(k)}$.

Similarly, the probability that the median is below $d/3$ is also $2^{-\Omega(k)}$.

Choosing $k = \Theta(\log(1/\delta))$, we can make the sum of these two probabilities work out to at most δ. This gives us a one-pass randomized streaming algorithm that computes an estimate $\hat{d}$ of d such that

$$\Pr[\hat{d} \notin [d/3, 3d]] \leq \delta.$$

What about the space requirement? The original algorithm requires $O(\log n)$ bits to store (and compute) the hash function and $O(\log \log n)$ bits to store z. Therefore, the space used by the final algorithm is $O(\log(1/\delta) \log n)$.