

Lecture 17: Finding Frequent Items, Distinct Elements

Notes by Ola Svensson¹

1 Estimating the Number of Distinct Elements

(This is the Facebook problem, i.e., the number of different cities on Facebook.)

DISTINCT-ELEMENTS problem Our goal is to output an approximation to the 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 $O(\log(1/\delta) \log n)$.

1.1 Ingredients

1.1.1 Pairwise independent hash family

A family \mathcal{H} of functions of the type $[n] \rightarrow [n]$ is said to be a pairwise independent hash family if the following property holds, with $h \in \mathcal{H}$ picked uniformly at random:

for any $x \neq x' \in [n]$ and $y, y' \in [n]$ we have

$$\Pr_{h \sim \mathcal{H}}[h(x) = y \wedge h(x') = y'] = 1/n^2.$$

Note that this implies that $\Pr_h[h(x) = y] = 1/n$. For us the following fact will be important (which follows from the construction in last lecture):

Lemma 1 *There exists a pairwise independent hash family so that h can be sampled by picking $O(\log n)$ random bits. Moreover, $h(x)$ can be calculated in space $O(\log n)$.*

1.1.2 The zero function

For an integer $p > 0$ let $\text{zeros}(p)$ denote the number of zeros that the binary representation of p ends with. Formally,

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

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

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 They have not been peer-reviewed and may contain inconsistent notation, typos, and omit citations of relevant works.

1.2 The Algorithm

The basic intuition of the algorithm is as follows:

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

The algorithm is now very simple:

Initialization: Choose a random hash function $h : [n] \rightarrow [n]$ from a pairwise independent family². Let $z = 0$.

Process j : If $\text{zeros}(h(j)) > z$ then $z = \text{zeros}(h(j))$.

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

We only use $O(\log n)$ space. Let's now analyze the quality of the output.

1.3 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 “ $\text{zeros}(h(j)) \geq r$,” and let $Y_r = \sum_{j:f_j>0} X_{r,j}$.
- Let t denote the value of z when algorithm terminates. By definition,

$$Y_r > 0 \iff t \geq r. \quad (1)$$

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

$$Y_r = 0 \iff t \leq r - 1. \quad (2)$$

In the next lecture we will compute the expectation and variance of Y_r , and use these bounds to prove that Y_r is likely positive for r such that $2^{r+1/2} < d/3$ and Y_{r+1} is likely zero for r such that $2^{r+1/2} > 3d$. This will establish the required approximation guarantee.

Let's now analyze the quality of the output.

1.4 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 “ $\text{zeros}(h(j)) \geq r$,” and let $Y_r = \sum_{j:f_j>0} X_{r,j}$.
- Let t denote the value of z when algorithm terminates. By definition,

$$Y_r > 0 \iff t \geq r. \quad (3)$$

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

$$Y_r = 0 \iff t \leq r - 1. \quad (4)$$

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

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

²To 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 $\log_2(n)$.

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

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

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

Then by using Markov's inequality, we have

$$\Pr[Y_r > 0] = \Pr[Y_r \geq 1] \leq \frac{\mathbb{E}[Y_r]}{1} = \frac{d}{2^r} \quad (5)$$

(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

$$\Pr[Y_r = 0] \leq \Pr\left[|Y_r - \mathbb{E}[Y_r]| \geq \frac{d}{2^r}\right] \leq \frac{\text{Var}[Y_r]}{(d/2^r)^2} \leq \frac{2^r}{d}. \quad (6)$$

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

- 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 (3) and (5),

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

- Similarly, let b the largest integer such that $2^{b+1/2} \leq d/3$. Using Equations (4) and (6),

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

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.

1.5 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 $O(\log(1/\delta) \log n)$.