Advanced Algorithms 11/28, 2017

Lecture 11

Lecturer: Mohsen Ghaffari *Scribe: Jerri Nummenpalo*

This draft has not been checked by the lecturer yet.

1 Streaming & Sketching Algorithms 1 — Frequent Elements, Approximate Counting, and Distinct Elements

1.1 Streaming algorithms

In the this lecture we will focus on streaming algorithms. This is a relatively new field of algorithm design which has seen many developments in the recent 5 years. We consider a *stream* of numbers $a_1, \dots, a_m \in \{0, \dots, n-1\}$ where the number are revealed to us one by one. The goal of a streaming algorithm is to take the stream and to maintain a function or an approximation of a function over the so far revealed stream. The main measure of complexity we are interested in is the amount of space the algorithm needs. Let us start with a simple example.

Consider the *Majority problem*: Given a stream $a_1, \dots, a_m$ with the guarantee that one element appears at least $\frac{m}{2}$ times, output such a majority element. An elegant algorithm for solving this problem maintains a pair (c, e) which we initialize after seeing the first element to $c \leftarrow 1$ and $e \leftarrow a_1$. We think of c as being a counter and e will always be one of the elements of the stream. Upon the arrival of a new element a_i we check if e equals a_i. If it does, we increase c by 1. If not, we check if $c > 0$ and in the positive case decrease c by 1 and in the negative case we set $e \leftarrow a_i$ and $c \leftarrow 1$. We claim that in the end e equals the majority element. This can be proven by pairing up a subset of the majority elements with the remaining elements (in a natural way) and then analyzing the behavior of the counter across the stream. The space needed by this algorithm is $\log n + \log m$. Notice that this algorithm only works for the promise version and it will not distinguish whether a majority element exists. In fact, in the exercises we will see that linearly many bits are needed to decide if a majority element exists.

1.2 Moment estimation

Many applications of streaming algorithms can be reduced to that of computing some *moment* of the stream. We let f_i denote the number of times that $i \in \{0, \dots, n-1\}$ appears in the stream. The *k-th moment* of the stream is defined as

$$F^k := \sum_{i=1}^{n} f_i^k. \tag{1}$$

1.2.1 First moment estimation

Let us consider the task approximating m in the stream up to a multiplicative factor of $(1 \pm \epsilon)$ for a constant $\epsilon > 0$. Equivalently, we want to compute a $(1 \pm \epsilon)$-approximation of the 1st moment of the stream. We consider the following approach due to Morris [Mor78] and uses $O(\log \log n)$ bits of memory. We initially set $x = 0$ and upon the arrival of a new element we increment x with probability 2^{-x}. In the end we output $2^x - 1$. For intuition, notice that if at some point $x = i$ it will take in expectation 2^i new elements until we increment x again. Since $m \approx \sum_{i=0}^{\log m} 2^i$ this strategy should give a good estimate of m.

For a more formal treatment we let X_i denote the value of x after we have seen i elements of the stream.

Claim 1. *The output of the algorithm is an unbiased estimator of m, i.e., $\mathbb{E}\left[2^{X_m}\right] = m + 1$.*

Proof. We prove the claim by induction on m. The base case $m = 0$ holds since $\mathbb{E}\left[2^0\right] = 0+1$. Assuming that the claim holds for some $m \geq 0$ we can condition on the value of X_m and compute that

$$\begin{aligned}\mathbb{E}\left[2^{X_{m+1}}\right] &= \sum_{j\geq 0} \mathbb{E}\left[2^{X_m} \mid X_m = j\right] \Pr[X_m = j] = \sum_{j\geq 0} \left(2^{-j}\cdot 2^{j+1} + (1-2^{-j})\cdot 2^j\right) \Pr[X_m = j] \\ &= \underbrace{\sum_{j\geq 0} \Pr[X_m = j]}_{=1} + \underbrace{\sum_{j\geq 0} 2^j \Pr[X_m = j]}_{=\mathbb{E}[2^{X_m}]} \overset{ind.}{=} 1 + (m+1).\end{aligned}$$

□

To bound the probability of the output 2^{X_m} deviating far from its expectation $m+1$, we use Markov's inequality to observe that

$$\Pr\left[|2^{X_m} - 1 - m| > \epsilon m\right] \leq \frac{\mathbb{E}[(2^{X_m} - 1 - m)^2]}{\epsilon^2 m^2} \leq \frac{1}{2\epsilon^2}. \tag{2}$$

The last inequality follows by first proving by induction along the similar lines as in Claim 1 that $\mathbb{E}[2^{2X_m}] = \frac{3}{2}m^2 + \frac{3}{2}m + 1$ and then via a direct computation we get that

$$\mathbb{E}[(2^{X_m} - 1 - m)^2] = \mathbb{E}[2^{2X_m}] - 2\mathbb{E}[2^{X_m}](m+1) + (m+1)^2 = \mathbb{E}[2^{2X_m}] - (m+1)^2 = \frac{m^2 - m}{2} \leq \frac{m^2}{2}.$$

Notice that for small ϵ the bound in (2) is trivial. To get a better bound observe first that $\mathbb{E}[(2^{X_m} - 1 - m)^2]$ is the variance of our output. Recall that the variance of the average of s i.i.d. random variables is a factor s smaller than the variance of only one such random variable. We can maintain $s = \frac{C}{\epsilon^2}$ independent counting processes x for some constant C and take the average of the resulting output. The probability that the result deviates more than ϵm from the expectation is then upper bounded by $\frac{1}{2\epsilon^2 s} = \frac{1}{2C}$.

If we want the error probability to be smaller than δ for some $\delta > 0$, then we could of course set $C = \frac{2}{\delta}$. A better idea is to fix C to be constant and, to maintain $O(\log \frac{1}{\delta})$ independent copies of the process, and to report the median of the outputs of the different processes. We leave it as an exercise to show that this gives the desired result. The number of bits used by this last process is $O\left(\frac{\log \frac{1}{\delta}}{\epsilon^2} \log\log n\right)$ and it gives an $(1 \pm \epsilon)$-approximation with probability $1 - \delta$.

1.2.2 Zeroth moment estimation

We go on to the estimation of the 0th moment of the stream, i.e., estimating the number of distinct elements in the stream. This section is based on the papers by Flajolet and Martin [FM85] and Alon, Matias and Szegedy [AMS96]. We let $\mathbb{Z}_n$ denote the set of integers $\{0, \ldots, n-1\}$. The basic idea is to take a random hash function $h : \mathbb{Z}_n \to [0, 1]$ and then to remember $\min\{h(a_i)\}$ across the stream. We output $\frac{1}{\min\{h(a_i)\}} - 1$ which is an unbiased estimator of the number of elements in the stream since the minimum of k uniformly distributed numbers in $[0, 1]$ is $\frac{1}{k+1}$ in expectation (exercise). Observe that this works because repeated elements of the stream are always mapped to the same value.

There are two shortcomings to this procedure that we haven't specified. The first problem is that we are hashing to real numbers. We can overcome this easily by considering instead hash functions $h : \mathbb{Z}_n \to \mathbb{Z}_n$. The second problem is the description of h. If we explicitly store a random hash function $h : \mathbb{Z}_n \to \mathbb{Z}_n$ we would have to use $\Theta(n \log n)$ bits of memory in expectation. This defeats the purpose of hashing as we might as well remember all the values we have seen in the stream in the same space. The answer to solving the second shortcoming is to only consider certain kinds of hash functions.

A family $\mathcal{H}$ consisting of hash functions $h : \mathbb{Z}_n \to \mathbb{Z}_n$ is a *family of pairwise independent hash functions* if for $h \in_{u.a.r.} \mathcal{H}$ and for any $x, y, i, i' \in \mathbb{Z}_n$ it holds that $\Pr[h(x) = i \wedge h(y) = i'] = \frac{1}{n^2}$. If n is a prime power, then $\mathbb{Z}_n$ equipped with standard addition and multiplication modulo n is a field. In that case $\mathcal{H}_n := \{h_{a,b} \mid a, b \in \mathbb{Z}_n \text{ and } h_{a,b}(x) = ax + b\}$ is a family of pairwise independent hash functions. This follows from the fact that for any $a, b, i, i' \in \mathbb{Z}_n$ the system $\{ax + b = i, ay + b = i'\}$ has a unique solution (x, y) in the field $\mathbb{Z}_n$. If n is not a prime power we can easily overcome this by instead considering the least power of 2 larger than n. We can store a random hash function from $\mathcal{H}_n$ by storing a and b which takes $O(\log n)$ bits. As a general remark, pairwise independence can be used for derandomization: If a randomized algorithm works correctly while using only pairwise independent randomness, then we can

brute force over all generators of that randomness which only leads to a polynomial multiplicative loss in the runtime.

Going back to the problem of estimating the 0th moment, for $i \in \mathbb{Z}_n$ we define zeros(i) as the largest integer j so that 2^j divides i. Our algorithm for the problem picks a random hash function $h_{a,b} \in \mathcal{H}_n$ and remembers $z := \max\{\text{zeros}(h_{a,b}(a_i))\}$ throughout the stream. We output $2^{z+\frac{1}{2}}$ in the end. The number of bits used for z is $O(\log \log n)$ so the total number of bits used is $O(\log n)$ due to having to store the hash function.

Claim 2. *The output $2^{z+\frac{1}{2}}$ is a 3-approximation to the number of distinct elements in the stream with constant probability.*

Proof. Consider the stream $a_1, \ldots, a_m$ and let S denote the set of distinct elements in the stream. For every $i \in S$ and for every $r = 1, \ldots, \log n$ we define the random variables

$$X_{i,r} := \begin{cases} 1 & \text{if zeros}(h_{a,b}(i)) \geq r \\ 0 & \text{otherwise} \end{cases}, \quad Y_r := \sum_{i \in S} X_{i,r}$$

and we let $d = |S| = F^0$ denote the number of distinct elements in the stream. Observe that $\mathbb{E}[X_{i,r}] = 2^{-r}$ since $X_{i,r} = 1$ iff the r least significant bits in the binary representation of $h_{a,b}(i)$ are 0. This happens with probability 2^{-r} because by construction $h_{a,b}(a_i)$ is uniformly distributed in $\mathbb{Z}_n$ and we can further assume that n is a power of two. From linearity of expectation we get that $\mathbb{E}[Y_r] = d2^{-r}$. The random variables Y_r are decreasing in r and z is equal to the largest r such that $Y_r \geq 1$. Letting d_1 be the smallest value with $2^{d_1+\frac{1}{2}} > 3d$ we can bound the probability that z is large with Markov's inequality:

$$\Pr[z \geq d_1] = \Pr[Y_{d_1} \geq 1] \leq \frac{d}{2^{d_1}} \leq \frac{\sqrt{2}}{3}.$$

For bounding the probability of erring to the other side we let d_2 be the largest value such that $2^{d_2+\frac{1}{2}} < \frac{d}{3}$. Then similarly to how we bounded the expression in (2) we get that

$$\Pr[z \leq d_2] = \Pr[Y_{d_2+1} = 0] \leq \Pr\left[\left|Y_{d_2+1} - \frac{d}{2^{d_2+1}}\right| \geq \frac{d}{2^{d_2+1}}\right] \leq \frac{\text{var}(Y_{d_2+1})}{\left(\frac{d}{2^{d_2+1}}\right)^2}. \tag{3}$$

Notice that Y_{d_2+1} is the sum of indicator variables and these indicator variables are pairwise independent by construction. The variance of pairwise independent random variables is the sum of the variances of these variables (note that this is where we use the properties of the hash functions). This allows us to derive the bound

$$\text{var}(Y_{d_2+1}) = \sum_{i \in S} \text{var}(X_{i,d_2+1}) = \sum_{i \in S} \mathbb{E}[X_{i,d_2+1}^2] - \mathbb{E}[X_{i,d_2+1}]^2 \leq \sum_{i \in S} \mathbb{E}[X_{i,d_2+1}] = \frac{d}{2^{d_2+1}} = \mathbb{E}[Y_{d_2+1}]$$

where we used the fact that X_{i,d_2+1} is an indicator random variable. For general use it is good to remember this fact that we can bound the variance of a sum of pairwise independent indicator random variables by the expectation of the same sum. Substituting our derivation to (3) we get that

$$\Pr[z \leq d_2] \leq \frac{2^{d_2+1}}{d} \leq \frac{\sqrt{2}}{3}.$$

In conclusion, we have seen that $\Pr[\frac{d}{3} \leq |2^{z+\frac{1}{2}} - d| \leq 3d] \geq 1 - \frac{2\sqrt{2}}{3} > 0$ meaning that with constant probability this approach produces a 3-approximation of d and standard probability boosting methods allow us to improve this probability higher. □

To improve on the approximation factor, let us instead take a random hash function h uniformly from $\mathcal{H}_N$ where $N = n^3$. This increase is needed to deal with rounding issues and it guarantees that with high probability there will be no collisions of the hashed values of $\mathbb{Z}_n$. Our algorithm remembers the t smallest values of $h(a_i)$ where $t = O(\frac{1}{\epsilon^2})$ for some $\epsilon > 0$. By letting z be the largest number that we remembered, the output of our algorithm is $\frac{tN}{z}$. The intuition here is that if the stream contains d distinct values, then the t-th smallest hashed value will be in expectation roughly $\frac{t}{d}N$.

Claim 3. *This modified algorithm produces a $(1 \pm \epsilon)$-approximation of the number of distinct elements d with constant probability.*

Proof (partial). Let $d_{out} = \frac{tN}{z}$ be the number the algorithm outputs and let $\tau := \frac{1}{1-\epsilon} \cdot \frac{tN}{d}$ be a threshold value. Denote by $h \in_{u.a.r.} \mathcal{H}_N$ the randomly chosen hash function and let S still denote the set of different values in the stream. For $i \in \mathbb{Z}_n$ let X_i be an indicator random variable for the event that $h(i) < \tau$ and let further $Y := \sum_{i \in S} X_i$. Then

$$\Pr[d_{out} \leq (1-\epsilon)d] = \Pr[z \geq \tau] = \Pr[Y < t].$$

Observe that $\mathbb{E}[Y] = d\frac{\tau}{N} = \frac{t}{1-\epsilon}$. For $\epsilon \leq \frac{1}{3}$ it holds that $(1+\frac{\epsilon}{2})t \leq \frac{t}{1-\epsilon} \leq (1+\frac{3}{2}\epsilon)t$. Using this we can compute that

$$\Pr[Y < t] \leq \Pr\left[|Y - \mathbb{E}[Y]| \geq \frac{\epsilon t}{2}\right] \leq \frac{\mathrm{var}(Y)}{\left(\frac{\epsilon t}{2}\right)^2} \leq \frac{\mathbb{E}[Y]}{\left(\frac{\epsilon t}{2}\right)^2} \leq \frac{(1+\frac{3}{2}\epsilon)t}{\left(\frac{\epsilon t}{2}\right)^2} \leq \frac{6}{C}$$

when we set $t = \frac{C}{\epsilon^2}$ and we can choose the constant C. We didn't cover in class the analysis of the probability $d_{out} \geq (1+\epsilon)d$. □

References

[AMS96] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 20–29. ACM, 1996.

[FM85] Philippe Flajolet and G Nigel Martin. Probabilistic counting algorithms for data base applications. *Journal of computer and system sciences*, 31(2):182–209, 1985.

[Mor78] Robert Morris. Counting large numbers of events in small registers. *Communications of the ACM*, 21(10):840–842, 1978.