# CS49/249 (Randomized Algorithms), Spring 2021 : Lecture 16

Topic: Streaming I : Frequency Estimation via Count-Min

*Disclaimer: These notes have not gone through scrutiny and in all probability contain errors. Please discuss in Piazza/email errors to deeparnab@dartmouth.edu*

---

- **Streaming Algorithms.** Imagine there is a huge corpus of data and you are allowed to make a serial pass over it. At any time, you can store information in your local memory, but the size of your memory is orders of magnitude smaller than the data itself. What problems on the data corpus can we still solve? This set of questions has led to the rich area of streaming algorithms. A whole course can, and indeed in Dartmouth is, taught on it. In the next few lectures, we will see some of the main highlights focusing on how randomization is used to solve these problems. For many of them, randomness is indeed *necessary*, but why that is the case is outside the scope of this course.

- A paradigmatic example of streaming problems is *frequency analysis*. Imagine the data corpus being a collection of *updates* on a universe of size $n$. Formally, let $U$ be a universe of $n$ items, and let $\mathbf{f}$ be an $n$-dimensional vector where $\mathbf{f}_i$ denotes the number of occurrences of the element $i$ in our data. Each "update" in the data looks of the form $(i, c)$ where $i \in [n]$ and $c$ is an integer. This is supposed to capture the intuition that we encounter $c$ copies of this element $i$. In that case, we set $\mathbf{f}_i \leftarrow \mathbf{f}_i + c$. After we make one pass through the data, which takes say $m$ updates, the final vector $\mathbf{f}$ will denote the *histogram* of the $n$ elements in the data set. Frequency analysis asks what all about this frequency vector can we capture, given that our working memory $s \ll \min(m, n)$?

  **Remark:** *From a first reading, one may think $c$ is a* non-negative *integer. But this may not necessarily be the case. For instance, if we think of these updates as "bank transactions", then there could be both deposits and withdrawals. The complexity of the problems do change depending on if $c \geq 0$ or if $c$ is general. The always non-negative model is often called the* insertion only *model or the* cash register *model. The other is called the* dynamic streaming *or* turnstile *model.*

- We will look at the following two kinds of problems in the next few lectures. One, would be estimating *each* frequency. So, for every $i \in [n]$, we would like to *estimate* $\mathbf{f}_i$ up to certain accuracy. We will see a couple of (pretty widely used in practice) algorithms for this.

  The second will be estimating some broader statistics called *frequency moments*. Given a parameter $k > 0$, define

  $$F_k := \sum_{i=1}^{n} \mathbf{f}_i^k$$

  This is called the $k$th frequency moment. Note that $F_1$ is particularly easy to calculate in the insertion only model (you see it, right?). We will see algorithms for estimating $F_2$ and also every $F_k$ for $k \geq 2$, and will also look at an algorithm for estimating $F_0$, which is defined to be the *number of distinct elements* in the stream. That is, the number of $i$'s for which $\mathbf{f}_i \geq 1$. To do so, we will utilize many of the tools we have covered so far in the course.

- **Estimating Frequencies via Count-Min.** We first work in the insertion only model. So for every update $(i, c)$, we assume $c \geq 0$. The algorithm we give is the COUNT-MIN algorithm[1] by Graham Cormode and S. Muthukrishnan.

  The main idea behind this algorithm is simple: we hash the universe $[n]$ into a smaller set $[k]$ using a hash function $h$ drawn from a universal hash family, and we work with the "estimated frequency" vector on $[k]$. That is, we maintain $k$ counters $C[1]$ to $C[k]$, and every time an update $(i, c)$ appears, we increment the counter corresponding to $h(i)$; $C[h(i)] \leftarrow C[h(i)] + c$. At the end of the stream, we estimate the frequency of $i \in [n]$ as

  $$\widehat{\mathbf{f}}_i := C[h(i)]$$

  Since we have assume $c \geq 0$, this estimate is clearly an overestimate; for any $i$ we have $\widehat{\mathbf{f}}_i \geq \mathbf{f}_i$ with probability 1. In fact, we precisely know how much the overestimate is by. It is

  $$\widehat{\mathbf{f}}_i = \mathbf{f}_i + \sum_{j \in [n], j \neq i, h(j)=h(i)} \mathbf{f}_j \tag{1}$$

  To see this, apart from $i$, note that every update involving $j$, with $h(j) = h(i)$, will increment the same counter $C[h(i)]$. And this increment will precisely be $\mathbf{f}_j$ "times". Therefore, we get that the expected value of the estimate (expectation over the randomness in the choice of the hash-function) is

  $$\mathop{\mathbf{Exp}}_{h}[\widehat{\mathbf{f}}_i] = \mathbf{f}_i + \sum_{j \neq i} \mathbf{Pr}[h(j) = h(i)] \cdot \mathbf{f}_j \quad \Rightarrow \quad \mathop{\mathbf{Exp}}_{h}[\widehat{\mathbf{f}}_i] \leq \mathbf{f}_i + \frac{1}{k} \sum_{j \neq i} \mathbf{f}_j \leq \mathbf{f}_i + \frac{1}{k} \cdot \|\mathbf{f}\|_1$$

  Note that this estimate is **not** an unbiased estimate. Nevertheless, one can get a "high-probability" estimate. Also, the final analysis of the "error term" would be additive and would be expressed in terms of an additive $\varepsilon \|\mathbf{f}\|_1$. Note if we divide by $\|\mathbf{f}\|$, then this would imply an additive $\varepsilon$-approximation to the *relative* frequency. For the main application described later, this is the kind of guarantee one needs.

  A simple Markov's inequality application gives

  $$\mathbf{Pr}\left[\widehat{\mathbf{f}}_i - \mathbf{f}_i \geq \frac{e}{k} \|\mathbf{f}\|_1\right] \leq \frac{1}{e} \tag{2}$$

  To boost this probability, and using the fact that $\widehat{\mathbf{f}}_i \geq \mathbf{f}_i$ always, we simply repeat $t$ times and take the minimum. In other words, instead of maintaining $k$ counters, we maintain $tk$ counters, and each of the $t$ counters uses an *independent* draw of a hash-function from the UHF. Thus, we obtain $t$ different estimates $\widehat{\mathbf{f}}_i^{(1)}, \ldots, \widehat{\mathbf{f}}_i^{(t)}$. Our final estimate is the **minimum**.

- *The Algorithm.*

[1] Graham Cormode and S. Muthukrishnan. *"An improved data stream summary: The count-min sketch and its applications."* Journal of Algorithms, 55:58–75, 2006.

1: **procedure** COUNT-MIN($\varepsilon, \delta$):
2: Let $H$ be a universal hash family with domain size $[n]$ and range $k = \lceil \frac{e}{\varepsilon} \rceil$.
3: Sample $t = \lceil \ln(1/\delta) \rceil$ hash functions $h_1, \ldots, h_t$ independently from $H$. ▷ *Each hash function is assumed to be stored in $O(1)$ space*
4: Maintain $kt$ counters $C_1[1:k], C_2[1:k], \ldots, C_t[1:k]$.
5:
6: **for** update $(i, c)$: **do** ▷ *Assume $c \geq 0$*
7: Increment $C_j[h_j(i)] \leftarrow C_j[h_j(i)] + c$ for $1 \leq j \leq t$.
8: Upon Query $i \in [n]$, **return** $\widehat{\mathbf{f}}_i := \min_{1 \leq j \leq t} C_j[h_j(i)]$ for all $1 \leq i \leq n$.

- *Analysis.*

**Theorem 1** (Count-Min Analysis.)**.** The total space usage of the COUNT-MIN algorithm is $O\left(\frac{1}{\varepsilon} \lg\left(\frac{1}{\delta}\right)\right)$ words. For any $1 \leq i \leq n$, we have

$$\mathbf{f}_i \leq \widehat{\mathbf{f}}_i \leq \mathbf{f}_i + \varepsilon \cdot \|\mathbf{f}\|_1 \tag{3}$$

with probability $\geq 1 - \delta$.

*Proof.* Firstly, note that for any $i \in [n]$, since *every* $C_j[h_j(i)] \geq \mathbf{f}_i$, we have $\widehat{\mathbf{f}}_i \geq \mathbf{f}_i$ as well. Now fix an $i \in [n]$. From (2), for every $1 \leq j \leq t$ we get that

$$\mathbf{Pr}[C_j[h_j(i)] \geq \mathbf{f}_i + \varepsilon \cdot \|\mathbf{f}\|_1] \leq \frac{1}{e}$$

The probability that the minimum overestimates, therefore is

$$\mathbf{Pr}[\widehat{\mathbf{f}}_i \geq \mathbf{f}_i + \varepsilon \cdot \|\mathbf{f}\|_1] \leq \frac{1}{e^t} \leq \delta$$

if $t = \lceil \ln(1/\delta) \rceil$. The space bound arises because there are $\lceil \ln(1/\delta) \rceil \cdot \lceil e/\varepsilon \rceil$ different hash functions and counters, each taking $O(1)$ space. □

**Remark:** *Note that for every $i$, there is a probability $\delta$ of error. The theorem is* **not** *stating that with probability $1-\delta$,* all *estimates are withing the desired range. To get every $i$ within the desired range, we would have to use union bound and the space would go up to $O(\varepsilon^{-1} \log(n/\delta))$.*

- **Application: Range Queries.** What if you were interested in the frequency of a certain "interval" of the items? For instance, suppose the elements were totally ordered as $\{1, 2, \ldots, n\}$, and at the end of the stream we wanted to answer queries of the form : for $1 \leq a < b \leq n$, what is $\sum_{a \leq i \leq b} \mathbf{f}_i$? Can we estimate this *sum* to within $\varepsilon \|\mathbf{f}\|$?

  Note that although we can estimate *every* point to within $\varepsilon' \|\mathbf{f}\|$, the sum may lead to an error of $\varepsilon'|b-a| \|\mathbf{f}\|$. So, if we wanted the sum to be within $\varepsilon \|\mathbf{f}\|$, we would have to set $\varepsilon'$ to be $\varepsilon/n$. And that would be lead to basically using $O(n)$ space. Can we do better? We can.

*Observation 1.* COUNT-MIN as described above is designed so as to estimate $\mathbf{f}_i$ for every individual $i \in [n]$. However, given any *partition* $A_1 \cup A_2 \cup \ldots \cup A_r = [n]$ of the $n$ elements, we can modify the algorithm such that with $O(\frac{1}{\varepsilon}\lg(1/\delta) \cdot \log r)$ bits we can ensure that for any $i \in [r]$ we can obtain an estimate

$$\mathbf{f}(A_i) \leq \widehat{\mathbf{f}}(A_i) \leq \mathbf{f}(A_i) + \varepsilon \|\mathbf{f}\|_1$$

We simply treat every element in $A_i$ as the same element and use hash functions from $[r]$ to $[k]$. Let's call this modification INTERVALCOUNTMIN.

*Observation 2.* Now we solve the range-query problem using a "binary-search" style idea. For simplicity, let $n$ be a perfect power of 2. Consider a binary tree with $\lg n$ levels, whose nodes correspond to intervals of $[1:n]$. The root is the interval $[1:n]$. Its two children are $[1:n/2]$ and $[n/2+1:n]$, and so-on-and-so-forth till we have the individual elements at the leaves. So, in general, these $2n-1$ intervals look like $[j \cdot \frac{n}{2^i} + 1 : (j+1) \cdot \frac{n}{2^i}]$ with $0 \leq i \leq \lg n$ and $0 \leq j \leq 2^i - 1$. These intervals are called *dyadic intervals*. Here is a cool fact which is key to the problem at hand : *any* interval $[a, b]$ can be partitioned into $\leq 2\lg n$ dyadic intervals. Do you see why?

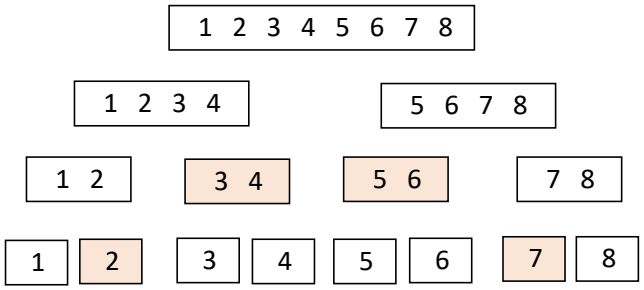


Figure 1: *An illustration of the dyadic intervals for* $n = 8$*. Note that the interval* $[2:7]$ *is broken down into* $4$ *dyadic intervals.*

Armed with this fact, here is what we do. At each layer of the tree, we have a partition of the $n$ elements into dyadic intervals. For each such partition, we run INTERVALCOUNTMIN with $(\frac{\varepsilon}{2\lg n}, \frac{\delta}{2\lg n})$. Given $[a, b]$, we first find $[a, b] = I_1 + \ldots + I_\ell$ for $\ell \leq 2\lg n$. By our INTERVALCOUNTMIN property, we have that with probability $1 - \ell \cdot \frac{\delta}{2\lg n} \geq 1 - \delta$ (after union bounding),

$$\text{For each } 1 \leq j \leq \ell,\ \mathbf{f}(I_j) \leq \widehat{\mathbf{f}}(I_j) \leq \mathbf{f}(I_j) + \frac{\varepsilon}{2\lg n}\|\mathbf{f}\|_1 \ \Rightarrow\ \mathbf{f}([a,b]) \leq \widehat{\mathbf{f}}([a,b]) \leq \mathbf{f}([a,b]) + \varepsilon\|\mathbf{f}\|_1$$

This gives the desired error per query. How much space did we use? Each INTERVALCOUNTMIN requires $O\left(\frac{\log n}{\varepsilon} \cdot \log\left(\frac{\lg n}{\delta}\right)\right)$, and there are $O(\log n)$ such data-structures. This gives a total space of $O(\frac{\log^2 n}{\varepsilon} \cdot \log(\frac{\log n}{\delta}))$.

**Exercise:** *Remove the* $\log\log n$ *term and show how you would modify the above to get a* $O(\frac{\log^2 n}{\varepsilon} \cdot \log(1/\delta))$ *dependency? Hint : Don't use black-box, but modify.*