CSE 290A: Randomized algorithms
Lecture 16: the Count-Min sketch, and the AMS algorithm for frequency moments

Lecturer: C. Seshadhri
Scribe: Zehui Cheng
University of California, Santa Cruz
Month May, 2020

1 Misra-Gries Algorithm

In the field of streaming algorithms, Misra-Gries summaries are used to solve the frequent elements problem about which (if any) value makes up a majority of the stream. In the paper [**?**], the authors proposed a deterministic algorithm which finds values occurring more than n/k times in an stream $[0 : n-1]$, and requires time proportional to $n \times \log(k)$ and extra space proportional to k. Furthermore, this paper proved that finding the frequent elements that appear more than $n/2$ times requires linear time. The first pass of this algorithm determines a set $\mathbf{t}$ of values that may occur more than n/k times in $[0 : n-1]$, while the second pass determine how many times each value in $\mathbf{t}$ which is in time $O(n \log(|\mathbf{t}|))$.

However, there are two issues with sampling and Misra-Gries Algorithm.

1. Algorithms are not SKETCHES which is defined as easily combinable (composable) summary of stream. Given streams A_1 and A_2, algorithms are SKETCHES if $sk(A_1 \circ A_2)$ can be easily computed from $sk(A_1)$ and $sk(A_2)$ ($\circ$ means concatenation). Figure 1 a sketch.

2. Algorithms work in cash register, not turnstile model. [**?**] and reservoir sampling do not handle deletions.

2 Count-Min Sketch

Count-min sketch (CM sketch) is a probabilistic data structure that serves as a frequency table. A set of hash functions are used to map events to frequencies [**?**]. Assume we have t independent hash functions ($1 \leq s \leq t$), such that $h_s : [n] \to [k]$. A matrix C can be built as shown in Figure 2.

1. Update: We update the matrix C for i when seeing a pair (i, c), such that $\forall s \leq t$, we have $C[s][h_s(i)] += c$.

2. Output: For all $j \in [n]$, $\hat{f}_j = \min_{s \leq t}(C[s][h_s(i)])$.

3. Storage: $kt \lg m$ (in realistic model) $+t \lg n$ (in theory, to get 2-universal hash functions).

It should be noted that for each hash function, there will be more than one value in $[1 : n]$ map into a same hash code. Therefore, $C[s][h_s(i)]$ records the occurring times of values in $[1 : n]$, whose hash code is $h_s(i)$. Then combining the sketches is logically equivalent to adding count matrices. Function $\min_{s \leq t}(C[s][h_s(i)])$ is used to approximate the frequency of j, because each cell in the matrix stores sum of frequencies of several values in $[n]$.

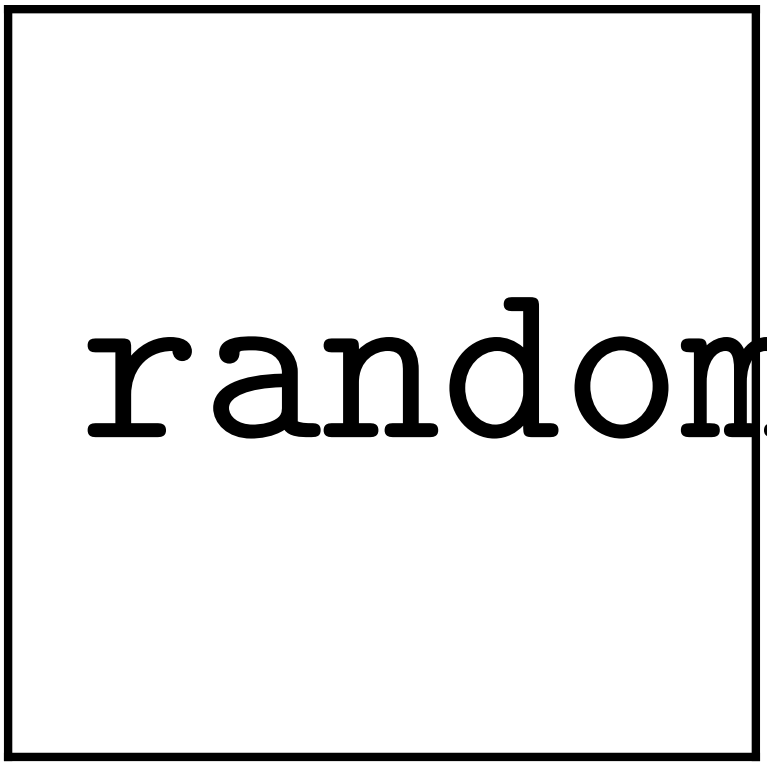

Figure 1: Memory 3 is computed from Memory 1 and Memory 2.

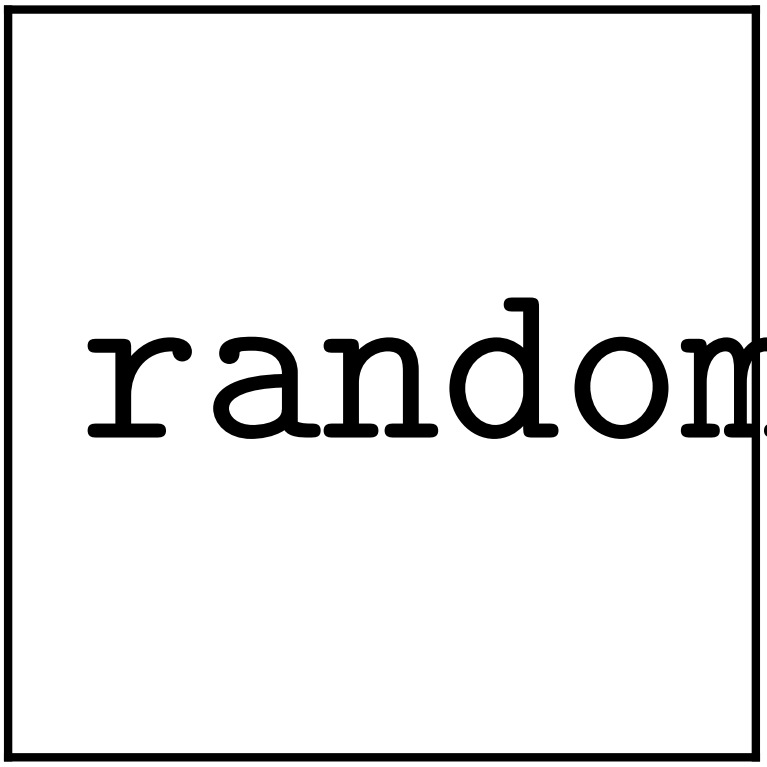

Figure 2: t hash functions.

Claim 2.1. *$\forall s \leq t C[s][h_s(j)] \geq f_j$ (in cash register model), and furthermore, $\hat{f}_j = \min_{s \leq t}(C[s][h_s(i)]) \leq f_j$.*

Proof. According to the definition of $h_s : [n] \to k (k < n)$, we know that there must be at least two values in $[n]$ fall into the same position (have the same hash code). This is also a birthday paradox. In addition, cash register model tells us that each position $h_s(j)$ is incremented by c when seeing (j, c). Therefore, we can say that for each position of row s ($\forall s \leq t$) in C, it records frequencies for at least one value in $[n]$. In the other word, in an arbitrary row s in C, for a value $j \in [n]$, we have $C[s][h_s(j)] \geq f_j$. Thus, $\min_{s \leq t}(C[s][h_s(i)]) \leq f_j$. □

It is natural to think how much the $\hat{f}_j - f_j$ is? we can guarantee that given a j, when $t \geq \lg(\frac{1}{\delta})$, we have $\hat{f}_j - f_j \leq \frac{||f||_1}{k} = \frac{m}{k}$ with probability $\geq 1 - \delta$. And thus the storage is $O(k \lg(\frac{1}{\delta}) \lg m)$ (realistic model).

Let $X_j = \hat{f}_j - f_j$ (overestimate of element j); and let $Y_{i,j}^{(s)}$ to be indicator for $h_s(i) = h_s(j)$; let $X_j^{(s)} = (C[s][h_s(i)]) - f_j$ (overestimate of element j in s^{th} hash function). Therefore, we have

$$X_j = \min_s X_j^{(s)}$$

$$X_j^{(s)} = \sum_{i \in [n] \wedge i \neq j} Y_{i,j}^{(s)} \times f_i \text{(Linearity of expectation)}$$

$$\begin{aligned} \mathbb{E}[X_j^{(s)}] &= \sum_{i \in [n] \wedge i \neq j} f_i \times \mathbb{E}[Y_{i,j}^{(s)}] \\ &\leq \frac{1}{k} \sum_{i \in [n]} f_i \\ &\leq \frac{||f||_1}{k} \end{aligned}$$

$\mathbb{P}[X_j^{(s)} \geq \frac{2||f||_1}{k}] \leq \frac{1}{2}$ (*by* Markov which is for any non-negative random variables)

Note that $\forall i \neq j \ \mathbb{P}[h_s(i) = h_s(j)] = \frac{1}{k}$ according to properties of 2-universal family.

Therefore,

$$\mathbb{P}[\textit{all overestimate are } \leq \frac{2||f||_1}{k}]$$

over independence of hash functions $h_1, \ldots, h_t$, is

$$\prod_{s \leq t} \mathbb{P}[X_j^{(s)} \geq \frac{2||f||_1}{k}] \leq \frac{1}{2^t} \leq \delta$$

where $t \geq \lg \frac{1}{\delta}$.

If we set $t = \lceil \lg \frac{1}{\delta} \rceil$, given a j, then $\mathbb{P}[\textit{final estimate} \geq f_j + \frac{2||f||_1}{k}] \leq \delta$. And the storage is in $O(k \lg(\frac{1}{\delta}) \lg m)$ bits ($m = ||f||_1$).

It should be noted that in turnstile model, we can use Count-Median. (output Median of $C[s][h_s(j)]$).

3 Estimating Frequency Moments

$F_k = \sum_{i \in [n]} f_i^k = ||\bar{f}||_k^k$. The second moment F_2 is of special interest. If we are streaming through the tuples of a database, then F_2 is the size of the self-join(#paths of length 2), where f_j is the number of tuples with value j of the join attribute.

A basic algorithm to estimate F_k ($k > 0$) is presented in paper [**?**]. Idea: suppose we could sample i proportional to f_i. And output $f_i^{k-1} \times m$ ($m = F_1 = ||\bar{f}||_1$). Then we have

$$\begin{aligned} \mathbb{E}[output] &= \sum_{i \in [n]} \frac{f_i}{m} \times m \times f_i^{k-1} \\ &= F_k \end{aligned}$$

By reservoir sampling, we get a random element at the END of the stream, so we cannot compute f_i a prior. On sampling i, we can only count SUBSEQUENT occurrences of i as shown in Figure 3. Conditional on sampling i, we want output to be $m \times f_i^{k-1}$. Suppose

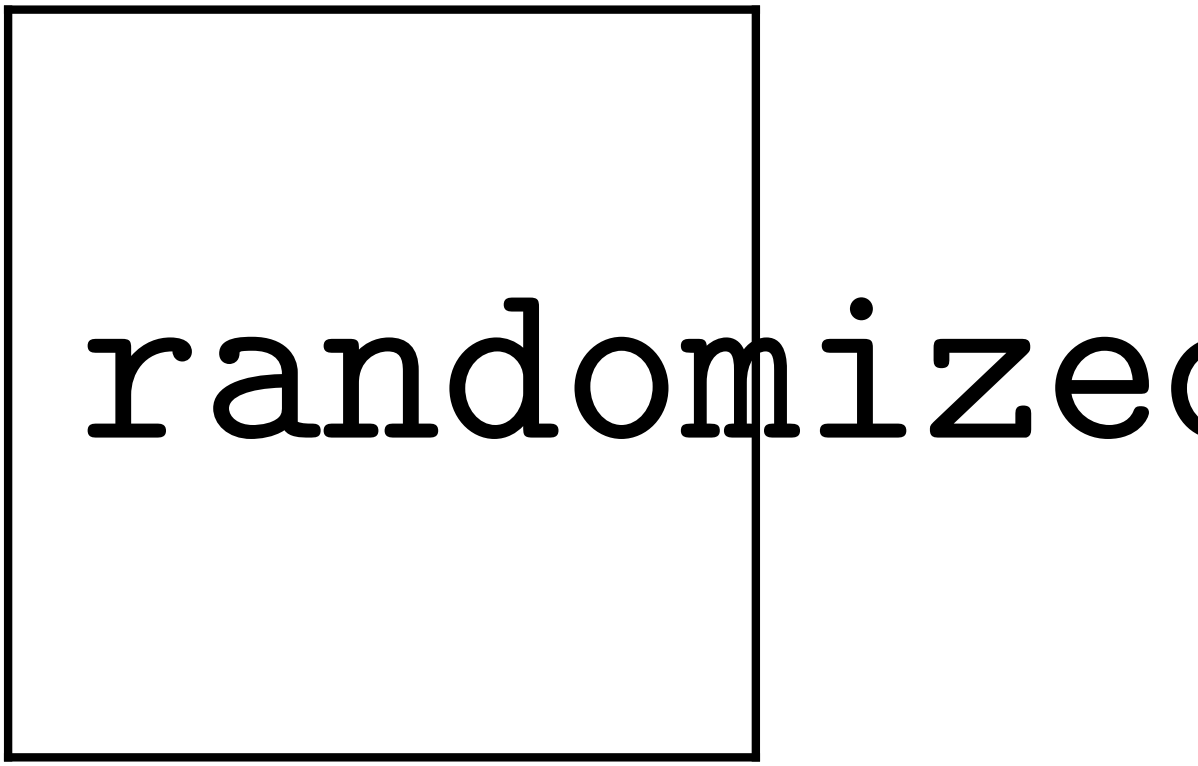

Figure 3: Reservoir

we output $m \times r^{k-1}$, where r is the number of subsequent occurrences.

We get (conditioned on sampling i) $\frac{1}{f_i} \sum_{s=1}^{f_i} s^{k-1} \approx \frac{1}{f_i} \times \frac{f_i^k}{k} \approx \frac{f_i^{k-1}}{k}$ (Cash register model), where s is the number of subsequent occurrences of i (not f_i), and i is uniformly picked from f_i by $\frac{1}{f_i}$.

In order to reduce the bias of estimation, a basic AMS algorithm is introduced. Run reservoir sampling. Keep track of number of subsequent occurrences of stored element (call

this r). Output $m(r^k - (r-1)^k)$. Then we have

$$\begin{aligned}
\mathbb{E}[output] &= \sum_{i=1}^{n} \mathbb{E}[output|i \text{ being reservoir sample }] \times \mathbb{P}[i \text{ is reservoir sample}] \\
&= \frac{1}{m} \times \sum_{i=1}^{n} f_i \times \mathbb{E}[output|i \text{ being reservoir sample }] \\
&= \frac{1}{m} \times \sum_{i=1}^{n} f_i \times \sum_{r=1}^{f_i} \frac{m}{f_i}(r^k - (r-1)^k) \\
&= \sum_{i=1}^{n} \sum_{r=1}^{f_i} (r^k - (r-1)^k) \\
&= \sum_{i=1}^{n} f_i^k = F_k
\end{aligned}$$

Theorem 3.1. $var[output] \leq n^{1-1/k}\mathbb{E}[output]^2$.

Eventually, this yields an $\widetilde{O}(\frac{n^{1-1/k}}{\epsilon^2})$ space algorithm. When $k = 2$, we have $\widetilde{O}(\frac{n^{1/2}}{\epsilon^2})$ (like birthday paradox). In fact, in classical AMS algorithm [**?**], it is $O(\frac{\log n}{\epsilon^2})$ space algorithm.

In the AMS algorithm [**?**] sketch for F_2, $||\bar{f}||_2^2$ is (squared) length of $\bar{f}$. Implicitly, each stream increment/decrement is an linear update to the vector $\bar{f}$. Then by use Johnson-Lindenstrauss lemma which helps with reduce high dimension into lower dimension, we can project $\bar{f}$ to $\frac{1}{\epsilon^2}$ dimensions, and length is preserved up to $(1 \pm \epsilon) - factor$ as shown in Figure 4. Store a random Gaussian vector $\bar{g} \in \mathbb{R}$ (each coordinate with $g \sim \mathcal{N}(0,1)$). On seeing element/pair in stream, update $\bar{f} \times \bar{g}$, such that update $X = x + c\bar{g}_j$. Output $X^2 = (\bar{f} \times \bar{g})^2$. Recall that $\bar{f} \times \bar{g}$ is a gaussian of mean 0 and variance $||\bar{f}||_2^2$.

A problem of JL y algorithm is storing $\bar{g}$ requires n space. A possible solution for this is that we don't need full independence 4-wise independent suffices. We don't really need gaussians. We just need $\mathbb{E}[g_i] = 0$ and $\mathbb{E}[g_i^2] = 1$.

Finally, we can choose $g : [n] \to \{-1, +1\}$ to be a hash function.

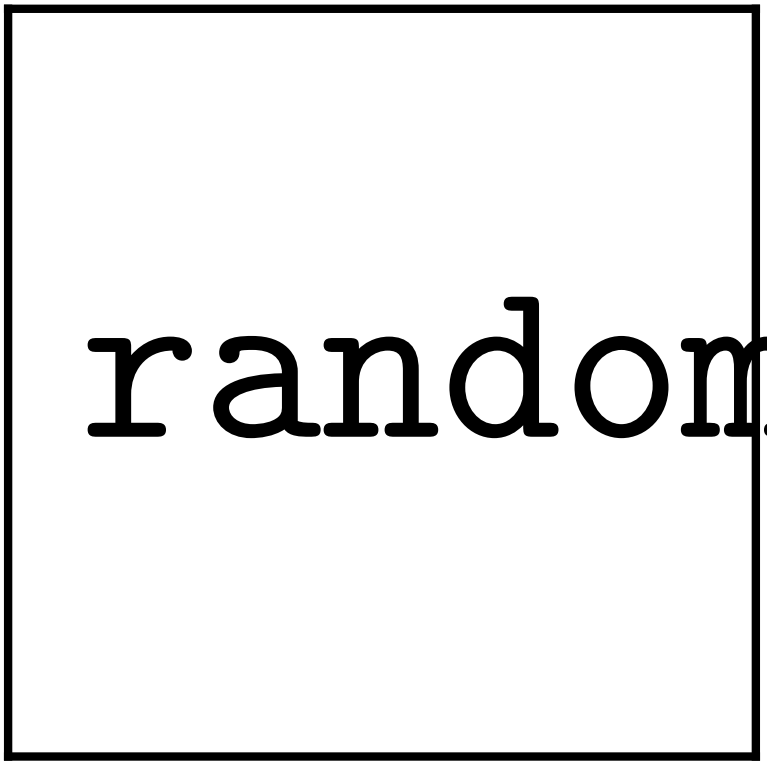

Figure 4: Dimension projection