Sketching Algorithms for Big Data Fall 2017

Lecture 2 — September 5, 2017

Prof. Jelani Nelson *Scribe: Saketh Rama*

1 Overview

In the last lecture, we surveyed the topics for this course, reviewed some probability theory, and considered Morris's algorithm for approximate counting with small registers.

In this lecture, we focus on another *streaming problem* – counting *distinct* elements in a stream. We consider an idealized solution to this problem, then outline the non-idealized solution which relies on k-wise independent hash functions.

2 Counting Distinct Elements in a Stream

More formally, consider a stream $i_1, i_2, i_3, \ldots, i_m \in \{1, \ldots, n\}$. We want our `query()` operation to return the number of distinct integers in the stream.

2.1 Trivial Solutions

The most obvious solution is to maintain a bitvector of length n (n bits). Another option is to just remember the stream ($\min(m, n) \lg n$ bits).

2.2 Goal for Today

We can actually solve this in $O(\min(m \lg n, n)$ bits of memory, but it turns out (though we will not prove this in lecture) that we need to require both approximate and randomized, else we need linear memory.

For today, we want to output some answer $\tilde{A}$ such that $\mathbb{P}(|\tilde{A} - \text{Answer}| > \varepsilon \cdot \text{Answer}) < \delta$. As usual, the ε is the approximation factor and the δ is the failure probability.

The first work to achieve this was [1]. We will do something similar, but not quite the same.

2.3 Idealized Solution

We will start with an idealized solution that uses real numbers – and therefore requires infinite memory! However, let's pretend that the real numbers don't require infinite memory for now, because the non-idealized solution is similar in spirit.

2.3.1 FM

Our basic algorithm (which we will call **FM** and subsequently upgrade into **FM+** and **FM++** as in previous lecture) proceeds as follows:

1. Pick random hash function $h : [n] \to [0, 1]$.
2. Maintain in memory the smallest hash we've seen so far: $X = \min_{i \in \text{stream}} h(i)$.
3. `query()`: output $1/X - 1$.

For some intuition, say that t is the number of distinct elements. We are partitioning the interval $[0, 1]$ into bins of size $1/(t + 1)$. With this in mind, we claim the following:

Claim 1. $\mathbb{E}[X] = \dfrac{1}{t+1}$.

Proof.

$$\begin{aligned}
\mathbb{E}[X] &= \int_0^\infty \mathbb{P}(X > \lambda)\, d\lambda \\
&= \int_0^\infty \mathbb{P}(\forall i \in \text{stream}, h(i) > \lambda)\, d\lambda \\
&= \int_0^\infty \prod_{i \in \text{stream}} \mathbb{P}(h(i) > \lambda)\, d\lambda \\
&= \int_0^1 (1 - \lambda)^t\, d\lambda \\
&= \frac{1}{t+1}
\end{aligned}$$

□

We compute similarly to get the variance $\text{Var}[X]$.

Claim 2. $\mathbb{E}[X^2] = \dfrac{2}{(t+1)(t+2)}$.

Proof.

$$\begin{aligned}
\mathbb{E}[X^2] &= \int_0^\infty \mathbb{P}(X^2 > \lambda)\, d\lambda \\
&= \int_0^\infty \mathbb{P}(X > \sqrt{\lambda})\, d\lambda \\
&= \int_0^1 (1 - \sqrt{\lambda})^t\, d\lambda \\
&= 2\int_0^1 u^t (1 - u)\, du \qquad [u = 1 - \sqrt{\lambda}] \qquad = \frac{2}{(t+1)(t+2)}
\end{aligned}$$

□

So

$$\mathrm{Var}[X] = \frac{2}{(t+1)(t+2)} - \frac{1}{(t+1)^2} = \frac{t}{(t+1)^2(t+2)} < (\mathbb{E}[X])^2.$$

2.3.2 FM+

We can upgrade our basic algorithm into **FM+** by running it $q = \frac{1}{\varepsilon^2 \eta}$ times in parallel to obtain $X_1, \ldots, X_q$. Then `query()` should output

$$\frac{1}{\frac{1}{q}\sum_{i=1}^{q} X_i} - 1.$$

Claim 3. $\mathbb{P}\left(\left|\frac{1}{q}\sum_{i=1}^{q} X_i - \frac{1}{t+1}\right| > \frac{\varepsilon}{t+1}\right) < \eta.$

Proof. By Chebyshev's inequality,

$$\begin{aligned}\mathbb{P}\left(\left|\frac{1}{q}\sum_{i=1}^{q} X_i - \frac{1}{t+1}\right| > \frac{\varepsilon}{t+1}\right) &< \frac{\mathrm{Var}[\frac{1}{q}\sum_i X_i]}{\frac{\varepsilon^2}{(t+1)^2}} \\ &< \frac{1}{\varepsilon^2 q} = \eta.\end{aligned}$$

□

This gives us a *linear* dependence on the failure probability, but we want *logarithmic*.

2.3.3 FM++

To achieve this, we will define **FM++** as running $t = \Theta(\lg \frac{1}{\delta})$ independent copies of FM+, each with $\eta = 1/3$. Then `query()` outputs the *median* across all FM+ estimates.

The new space for FM++ is now $O\left(\frac{1}{\epsilon^2} \lg \frac{1}{\delta}\right)$.

To analyze this, define indicator random variables $Y_1, \ldots, Y_t$, where Y_i is 1 iff the ith copy of FM+ failed to achieve a $(1+\epsilon)$-approximation (the event in the probability bound).

2.4 Non-Idealized Solution

First, we need a pseudorandom hash function h. We will use **k-wise independent hash functions**.

2.4.1 k-wise independent hash functions

Definition 4. *A family $\mathcal{H}$ of functions mapping $[a]$ into $[b]$ is k-*wise independent *iff for all distinct $i_1, \ldots, i_k \in [a]$ and for all $j_1, \ldots, j_k \in [b]$,*

$$\underset{h \in \mathcal{H}}{\mathbb{P}}(h(i_1) = j_1 \wedge \cdots \wedge h(i_k) = j_k) = \frac{1}{b^k}.$$

Note that we can store $h \in \mathcal{H}$ in memory with $\log_2 |\mathcal{H}|$ bits.

One example of such a family $\mathcal{H}$ is the set of all functions mapping $[a]$ to $[b]$. Then $|\mathcal{H}| = b^a$, and so $\lg |\mathcal{H}| = a \lg b$. A less trivial example is due to Carter and Wegman [2], where $\mathcal{H}$ is the set of all degree-$(k-1)$ polynomials over $\mathbb{F}_q$ such that $a = b = q$. Then $|\mathcal{H}| = q^k$, and so $\lg |\mathcal{H}| = k \lg q$. (This is not too hard to justify but we will not do so in lecture.)

Having seen these examples, we will just assume that we have access to some 2-wise independent hash families, which will let us store in $\lg n$ bits.

2.4.2 Common Strategy: Geometric Sampling of Streams

Suppose we have a substitute that gives us $\tilde{t}$ as a 32-approximation to t. To get the $(1+\varepsilon)$-approximation, we will use the common strategy of ***geometric sampling of streams***. This is important to understand because it is used fairly often in scenarios like this one.

First, let us consider the trivial solution (**TS**): remember the first K distinct elements in the stream, with $K = c/\varepsilon^2$. Our algorithm then composes these trivial solutions as follows:

1. Assume n is a power of 2.
2. Pick $g : [n] \to [n]$ from a 2-wise family.
3. `init()`: create $\lg n + 1$ trivial solutions $\text{TS}_0, \ldots, \text{TS}_K$.
4. `update(`i`)`: feed i to $\text{TS}_{\text{LSB}(g(i))}$
5. `query()`: choose j such that $\frac{\tilde{t}}{2^{j+1}} \approx \frac{1}{\epsilon^2}$. (We want this squared term due to Chebyshev's inequality.)
6. output: $\text{TS}_{j \cdot \texttt{query()} \cdot 2^{j+1}}$

Consider $g : [16] \to [16]$, say with $g(i) = 1010$. In this case, the LSB index is 1 (hence the "+1" in `init()`). For the LSB index to equal j, we need a "run" of $j-1$ zeros from right to left.

Define a set of *virtual streams* wrapping the trivial solutions.

$$\begin{array}{llcc} \text{VS} & 0 & \longrightarrow & \text{TS}_0 \\ \vdots & \vdots & \vdots & \vdots \\ \text{VS} & \lg n & \longrightarrow & \text{TS}_{\lg n} \end{array}$$

Fix some $j \in \{0, \dots, \lg n\}$. Let Z_i be an indicator random variable for $\text{LSB}(g(i)) = j$. Then the number of distinct elements in VS j is $\sum_{i \in \text{stream}} Z_i = Z$. Note that $\mathbb{E}[Z] = \frac{t}{2^{j+1}}$, and $\text{Var}[\sum_i Z_i] = \sum_i \text{Var}[Z_i]$ due to the pairwise independence we have inherited from our 2-wise hash function. (In fact, that is why we required the 2-wise independence in the first place, so that we can do this with the variance later on.) We can then conclude that

$$\sum_i \text{Var}[Z_i] < \frac{t}{2^{j+1}} = Q_j,$$

where we have denoted this last term as Q_j.

Now we can apply Chebyshev with a 9/10 probability. Note that $Z - Q_j = O(\sqrt{Q_j})$, so

$$Z = Q_j \pm O(\sqrt{Q_j}) = \left(1 \pm O\left(\frac{1}{\sqrt{Q_j}}\right)\right) Q_j.$$

We want the term inside the O-expression to be ε. (Also, if j is too small, such that $\tilde{t}/(2^{j+1})$ cannot be approximately $1/\varepsilon^2$ as needed, then just run the trivial solution alone and backoff to the above algorithm if needed.)

Without being too pedantic here, just find the highest nonempty virtual stream. We can analyze this to obtain the 90% probability stated at the outset.

How much space do we need? We need to store g ($\lg n$), and also TS_j for $j \in \{0, \dots, \lg n\}$. So in total, we need $O\left(\frac{1}{\varepsilon^2} \cdot \lg^2 n \cdot \lg \frac{1}{\delta}\right)$ *bits*.

2.5 State-of-the-Art Bounds in Literature

2.5.1 Lower Bound

The lower bound is $\Omega\left(\frac{1}{\varepsilon^2} \lg \frac{1}{\delta} + \lg n\right)$ bits. For those interested in the history of this lower bound, see the following references:

1. [3]
2. [5]
3. [4]
4. [6]

2.5.2 Upper Bound

First was the work on "HyperLogLog", which established

$$O\left(\frac{1}{\varepsilon^2} \lg \lg n + \lg n\right).$$

Forthcoming work from Błasiok (2018?) has established

$$O\left(\frac{1}{\varepsilon^2}\lg\frac{1}{\delta^2}+\lg n\right),$$

and so the problem is pretty much completely solved.

3 Future Work: Continuous Monitoring

A related area which may offer interesting opportunities is the *continuous monitoring* problem, where we expect m queries and must maintain correctness throughout A basic union bound is a trivial solution to this.

References

[1] Philippe Flajolet, G. Nigel Martin. Probabilistic Counting Algorithms for Data Base Applications. *J. Comput. Syst. Sci.*, 31(2):182–209, 1985.

[2] J. Lawrence Carter, Mark N. Wegman. Universal Classes of Hash Functions. *Proceedings of the Ninth Annual ACM Symposium on Theory of Computing*, pp. 106–112, 1997.

[3] Noga Alon, Yossi Matias, Mario Szegedy. The Space Complexity of Approximating the Frequency Moments. *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, pp. 20–29, 1996.

[4] Piotr Indyk, David Woodruff. Optimal Approximations of the Frequency Moments of Data Streams. *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing*, pp. 202–208, 2005.

[5] David Woodruff. Optimal Space Lower Bounds for All Frequency Moments. *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 167–175, 2004.

[6] Thathachar Jayram, David Woodruff. The Data Stream Space Complexity of Cascaded Norms. *50th Annual IEEE Symposium on Foundations of Computer Science*, pp. 765–774, 2009.

[7] Philippe Flajolet, Éric Fusy, Olivier Gandouet, Frédéric Meunier. Hyperloglog: The Analysis of a Near-Optimal Cardinality Estimation Algorithm. *AofA: Analysis of Algorithms*, pp. 137–156, 2007.