Algorithms for Massive Data Lecture Notes

Darshan Thaker

February 6, 2019

Contents

Lecture 1

Approximate Counting

We consider counting up to $n \in \mathbb{N}$ in the streaming model. To motivate this, consider a router that receives many requests and wants to count the number of requests. Naively, to store a counter for n, this would require $O(\log(n))$ bits of space (the big-O arises from the chosen base of the logarithm, which can be absorbed into a constant). Can we do better? The answer is a resounding no, *unless* we allow for approximation and randomness. If we allow approximation, the surprising answer is that we can use only $O(\log\log(n))$ bits of storage.

1.1 Morris's Algorithm

Algorithm 1.1, known as Morris's algorithm, is such an approximate algorithm.

Algorithm 1.1 Approximate Counting Algorithm [Morris 78]

```
Given: a stream of data.
  Initialize variable $X = 0$.
  Upon each new element in the stream, increment $X$ with probability $\frac{1}{2^X}$, else do nothing.
  Return estimator $E = 2^X - 1$.
```

We now analyze the correctness and performance of the Morris algorithm. First, let us start with the correctness of the algorithm. Let X_n be the value of X after n increments, and let $E_n = 2^{X_n} - 1$.

Theorem 1.1.1. *The estimator outputted by the Morris algorithm is a unbiased estimator, i.e.*

$$\mathbb{E}_{X_1,\ldots,X_n}[E_n] = n$$

Proof. The proof proceeds by induction over n. Note that the expectation is over the n values of X (since these are all random variables). First, note the base case of $n = 0$. Clearly, $E_0 = 2^{X_0} - 1 = 2^0 - 1 = 0$, thus the base case is satisfied. Now, assume that the hypothesis holds for E_{n-1}, and we will show that it holds also for E_n. First, we will compute $\mathbb{E}_{X_1,\ldots,X_n}[2^{X_n}]$. We can split this expectation into two expectations:

$$\mathbb{E}_{X_1,\ldots,X_n}[2^{X_n}] = \mathbb{E}_{X_1,\ldots,X_{n-1}}\left[\mathbb{E}_{X_n}\left[2^{X_n}\right]\right]$$

This trick is an important trick. It amounts to fixing the first $n-1$ draws of $X_1, \dots, X_{n-1}$ and then finding the expectation with respect to the last draw X_n. We can always do this by the rules of probability.

$$\begin{aligned}
&= \mathbb{E}_{X_1,\dots,X_{n-1}}\left[2^{X_{n-1}+1} \cdot \Pr\left[X_{n-1} \text{ incremented}\right] + 2^{X_{n-1}} \cdot \Pr\left[X_{n-1} \text{ not incremented}\right]\right] \\
&= \mathbb{E}_{X_1,\dots,X_{n-1}}\left[2^{X_{n-1}+1} \cdot \frac{1}{2^{X_{n-1}}} + 2^{X_{n-1}} \cdot \left(1 - \frac{1}{2^{X_{n-1}}}\right)\right] \\
&= \mathbb{E}_{X_1,\dots,X_{n-1}}\left[2 + 2^{X_{n-1}} - 1\right] \\
&= \mathbb{E}_{X_1,\dots,X_{n-1}}\left[2^{X_{n-1}}\right] + 1 \\
&= \mathbb{E}_{X_1,\dots,X_{n-1}}\left[E_{n-1} + 1\right] + 1 \\
&= \mathbb{E}_{X_1,\dots,X_{n-1}}\left[n - 1 + 1\right] + 1 && \text{(Inductive hypothesis)} \\
&= n + 1
\end{aligned}$$

Thus, by linearity of expectation (which we used above as well), we have that $\mathbb{E}_{X_1,\dots,X_n}[E_n] = \mathbb{E}_{X_1,\dots,X_n}[2^{X_n}] - 1 = n$, completing the proof. □

We are not merely interested in the expectation of the estimator, but also in the concentration of measure of the estimator, which we now prove.

Theorem 1.1.2. *The estimator outputted by the Morris algorithm has the following property.*

$$\operatorname{Var}[E_n] \le \frac{3n(n-1)}{2} + 1 = O(n^2)$$

Proof. First, we note that instead of proving results about $\operatorname{Var}[E_n]$, we can prove results about $\mathbb{E}[2^{2X_n}]$. To see this, note the following (where all expectations are taken over the random $X_1, \dots, X_n$):

$$\begin{aligned}
\operatorname{Var}\left[E_n\right] &= \operatorname{Var}\left[2^{X_n} - 1\right] \\
&= \operatorname{Var}\left[2^{X_n}\right] \\
&= \mathbb{E}\left[\left(2^{X_n}\right)^2\right] - \left(\mathbb{E}\left[2^{X_n}\right]\right)^2 \\
&\le \mathbb{E}\left[2^{2X_n}\right]
\end{aligned}$$

We now proceed via induction on n to show that $\mathbb{E}\left[2^{2X_n}\right] \le \frac{3n(n-1)}{2} + 1$. Consider the base case for $n = 0$. We know that $\mathbb{E}\left[2^{2X_0}\right] = 1$ based on the initialization of the algorithm, which is less than or equal to $\frac{3 \cdot 0 \cdot (0-1)}{2} + 1$. Now, assume that the hypothesis holds for $\mathbb{E}_{X_1,\dots,X_{n-1}}\left[2^{2X_{n-1}}\right]$. Following the proof almost exactly as Theorem 1.1.1, we can conclude that:

$$\begin{aligned}
\mathbb{E}_{X_1,\ldots,X_n}[2^{2X_n}] &= \mathbb{E}_{X_1,\ldots,X_{n-1}}\left[\mathbb{E}_{X_n}\left[2^{2X_n}\right]\right] \\
&= \mathbb{E}_{X_1,\ldots,X_{n-1}}\left[2^{2(X_{n-1}+1)}\cdot\frac{1}{2^{X_{n-1}}} + 2^{2X_{n-1}}\cdot\left(1-\frac{1}{2^{X_{n-1}}}\right)\right] \\
&= \mathbb{E}_{X_1,\ldots,X_{n-1}}\left[2^{X_{n-1}+2} + 2^{2X_{n-1}} - 2^{X_{n-1}}\right] \\
&= \mathbb{E}_{X_1,\ldots,X_{n-1}}\left[4\cdot 2^{X_{n-1}} + 2^{2X_{n-1}} - 2^{X_{n-1}}\right] \\
&= \mathbb{E}_{X_1,\ldots,X_{n-1}}\left[3\cdot 2^{X_{n-1}} + 2^{2X_{n-1}}\right] \\
&= 3\cdot\mathbb{E}_{X_1,\ldots,X_{n-1}}\left[2^{X_{n-1}}\right] + \mathbb{E}_{X_1,\ldots,X_{n-1}}\left[2^{2X_{n-1}}\right] \\
&= 3\cdot\mathbb{E}_{X_1,\ldots,X_{n-1}}\left[E_{n-1}+1\right] + \mathbb{E}_{X_1,\ldots,X_{n-1}}\left[2^{2X_{n-1}}\right] \\
&= 3n + \mathbb{E}_{X_1,\ldots,X_{n-1}}\left[2^{2X_{n-1}}\right] && \text{(Theorem 1.1.1)} \\
&\leq 3n + \frac{3(n-1)(n-2)}{2} && \text{(Inductive hypothesis)} \\
&= \frac{3n(n-1)}{2} + 1 \\
&= O(n^2)
\end{aligned}$$

By the principle of mathematical induction, we can conclude the claim holds for all natural numbers n. □

Combining the above two theorems above, we can now state a probabilistic result about the approximation error of the estimator outputted by Morris's algorithm. Before stating this result, recall Chebyshev's inequality (which is reviewed in the Appendix).

Theorem 1.1.3. *The estimator outputted by Morris's algorithm E_n is in the range $n \pm O(n)$ with probability at least* 90%.

Proof. We use Chebyshev's inequality on the random variable E_n with the value of $\lambda^2 = 10\text{Var}[E_n]$. By Theorem 1.1.1 and Theorem 1.1.2, we know that $\mathbb{E}[E_n] = n$ and $\text{Var}[E_n] = O(n^2)$. Thus, by the Chebyshev bound, we have that E_n falls in the range $\mathbb{E}[E_n] \pm \lambda$ with probability at least 90%, or that E_n falls in the range $n \pm O(n)$ with probability at least 90%. □

Finally, we observe that the Morris algorithm uses $O(\log\log(n))$ bits of storage with high probability, since the algorithm simply stores X and returns $E = 2^X - 1$. Since with high probability, E is "close" to n by the above theorem, we conclude that X requires $\log\log(n)$ bits of storage with high probability (at least 90%).

1.2 Variance Improvement by Averaging

We can improve the variance of the estimator outputted by the Morris algorithm using a generic technique. This leads to the so-called Morris+ algorithm, described in Algorithm 1.2, which simply outputs the average of k independent runs of the Morris algorithm.

Algorithm 1.2 Approximate Counting Algorithm with Better Variance [Morris 78]

```
Given: a stream of data, $k \in \mathbb{N}$
  Run $k$ independent copies of the Morris algorithm (Algorithm 1.1) to get $k$ estimators, $E_1, \ldots E_k$.
  Return estimator $E = \frac{1}{k}\sum_{i=1}^{k} E_i$.
```

We now analyze the correctness of this algorithm (using the same notation as the previous section).

Theorem 1.2.1. *The estimator outputted by the Morris+ algorithm is a unbiased estimator, i.e.*

$$\mathbb{E}_{E_1,\ldots,E_n}[E] = n$$

Proof. This follows easily from the linearity of expectation and the application of Theorem 1.1.1.

$$\begin{aligned}
\mathbb{E}_{E_1,\ldots,E_n}[E] &= \frac{1}{k}\sum_{i=1}^{k}\mathbb{E}_{E_i}[E_i] \\
&= \frac{1}{k}\cdot kn \\
&= n
\end{aligned}$$

□

Similarly, we can analyze the variance of the estimator.

Theorem 1.2.2. *The estimator outputted by the Morris+ algorithm has the following property (for $n \geq 10$).*

$$\mathrm{Var}[E] \leq \frac{3}{2}\cdot\frac{n^2}{k} = O\left(\frac{n^2}{k}\right)$$

Proof. By properties of the variance and the application of Theorem 1.1.2, we know the following:

$$\begin{aligned}
\mathrm{Var}\left[\frac{1}{k}\sum_{i=1}^{k}E_i\right] &= \frac{1}{k^2}\sum_{i=1}^{k}\mathrm{Var}[E_i] \\
&\leq \frac{1}{k^2}\cdot k\cdot\frac{3}{2}n^2 && \text{(If } n \geq 10\text{)} \\
&= \frac{3}{2}\cdot\frac{n^2}{k} \\
&= O\left(\frac{n^2}{k}\right)
\end{aligned}$$

□

Notice that the above theorem introduces an inverse dependence of the variance on k (which is a parameter we can pick). Thus, we can pick k to reduce the variance of the estimator as much as we desire. In particular, suppose we wish to achieve a $1+\epsilon$ approximation to the desired output with high probability, i.e. the outputted estimator should be in the range $[n,(1+\epsilon)n]$ with high probability. We can now state a result about this approximation using the Morris+ algorithm.

Theorem 1.2.3. *Using $k = O\left(\frac{1}{\epsilon^2}\right)$, the estimator outputted by the Morris+ algorithm E is in the range $n \pm \epsilon\cdot O(n)$ with probability at least 90%.*

Proof. We use Chebyshev's inequality on the random variable E with the value of $\lambda^2 = 10\mathrm{Var}[E]$. By Theorem 1.2.1 and Theorem 1.2.2, we know that $\mathbb{E}[E] = n$ and $\mathrm{Var}[E_n] = O\left(\frac{n^2}{k}\right)$. Thus, by the Chebyshev bound, we have that E falls in the range $\mathbb{E}[E] \pm \lambda$ with probability at least 90%, or that E falls in the range $n \pm \frac{1}{\sqrt{k}}O(n)$ with probability at least 90%. If $k = O\left(\frac{1}{\epsilon^2}\right)$, we have that E falls in the range $n \pm \epsilon\cdot O(n)$ with probability at least 90%. □

1.3 Median Trick

Observe that in the analysis in the previous sections, we used the Chebyshev bound to obtain a fixed confidence, i.e. with probability at least 90%, our estimator was ϵ-approximate. However, in many settings, we are interested in obtaining a specific confidence level via a parameter δ. Thus, with probability at least $1-\delta$, we want our estimator to be ϵ-approximate (this should ring a bell with PAC learning). How can we boost our Morris algorithm to arbitrary confidence, given the Morris algorithm that works with 90% confidence? One solution is to use a Chebyshev bound with λ depending upon δ. We picked $\lambda^2 = 10\text{Var}[X]$ to achieve a 90% confidence. More generally, we can pick $\lambda^2 = \frac{1}{\delta}$, which gives $k = O\left(\frac{1}{\epsilon^2}\frac{1}{\delta}\right)$ for the number of times to run the Morris algorithm to achieve a confidence level of at least $1-\delta$. If δ is small enough, this dependence upon $\frac{1}{\delta}$ can be quite prohibitive, so a natural question is: can we do better? It turns out we can!

The answer is to use the *median trick*, which we will describe in detail below. Given an algorithm A that has a certain confidence level, or a specific probability that the output of the algorithm is in a correct range, we can apply the median trick to create an algorithm A' such that the probability that the output of the algorithm is in a correct range is at least $1-\delta$. Additionally, we can avoid the $\frac{1}{\delta}$ dependence as in the solution above.

Algorithm 1.3 shows the median trick pseudocode. The algorithm is simple: simply run algorithm A k times and output the median of the estimators. Picking k wisely yields a very simple way to boost confidence.

Algorithm 1.3 Median Trick

```
Given: Algorithm $A$ with a certain confidence level, $k \in \mathbb{N}$
  Run $k$ independent copies of Algorithm $A$ to get $k$ estimators, $E_1, \dots E_k$.
  Return estimator $E$ which is the median of $E_1, \dots, E_k$.
```

Before we proceed with the analysis, remind yourselves of the Chernoff and Hoeffding bounds (reviewed in the appendices). We are now ready to prove a result about the median trick. Note that the median trick intuitively will only work for boosting confidence levels that are strictly greater than 0.5. This should make sense because a confidence level of 0.5 can be achieved by random guessing, so it should make sense that we cannot extract any useful information from random guessing.

Theorem 1.3.1. *If we take $k = O\left(\log\frac{1}{\delta}\right)$, the median trick will boost the confidence of Algorithm A from a confidence level c to at least $1-\delta$.*

Proof. Let $A_1, \dots, A_k$ be the output of Algorithm A on the k instances it is run. Let X_i be an indicator variable that is 1 when A_i is "correct" (in the desired approximate range), and 0 otherwise. Let $\mu = \mathbb{E}\left[\sum_{i=1}^k X_i\right]$. Then, $\mathbb{E}[X_i] \geq \Pr[X_i = 1] = c$ (because c is the confidence level of Algorithm A). Thus, we conclude that $\mu \geq c \cdot k$ (by independence). Note that Algorithm A' is correct if greater than half of the k estimators are correct, since the outputted value is the median. Note that this is a sufficient, not necessary condition. Because each X_i is an indicator variable, this condition is equivalent to the condition that $\sum_{i=1}^k X_i \geq 0.5k$. We would like the probability of this event to be at least $1-\delta$. Instead, we solve the easier problem of solving for k such that the probability (of failure) that $\sum_{i=1}^k X_i < 0.5k$ is upper bounded by δ. Observe the algebraic manipulation of this condition below to allow us to use the Chernoff bound:

$$\begin{aligned}
\Pr\left[\sum_{i=1}^{k} X_i < 0.5k\right] &= \Pr\left[\sum_{i=1}^{k} X_i - ck < (0.5-c)\cdot k\right] \\
&\le \Pr\left[\left|\sum_{i=1}^{k} X_i - ck\right| > (c-0.5)\cdot k\right] && (c > 0.5) \\
&\le \Pr\left[\left|\sum_{i=1}^{k} X_i - \mu\right| > \frac{c-0.5}{c}\cdot \mu\right] \\
&\le 2\exp\left\{-\frac{1}{3}\left(\frac{c-0.5}{c}\right)^2 \mu\right\} && (\text{Chernoff bound}) \\
&\le 2\exp\left\{-\frac{1}{3}\left(\frac{c-0.5}{c}\right)^2 ck\right\} && (\mu \ge c\dot{k})
\end{aligned}$$

We set this probability to be less than δ and solve for a suitable value of k:

$$\begin{aligned}
2\exp\left\{-\frac{1}{3}\left(\frac{c-0.5}{c}\right)^2 ck\right\} &< \delta \\
\frac{1}{3}\frac{(c-0.5)^2}{c}k &> \log\left(\frac{2}{\delta}\right) \\
k &> 3\frac{c}{(c-0.5)^2}\log\left(\frac{2}{\delta}\right) \\
k &> O\left(\log\left(\frac{1}{\delta}\right)\right)
\end{aligned}$$

□

With the median trick, we can improve Morris's algorithm to use $k = O\left(\frac{1}{\epsilon^2}\log\frac{1}{\delta}\right)$ to achieve an ϵ-approximate output with confidence $1-\delta$. The algorithm runs the Morris+ algorithm first to achieve a fixed confidence level and then uses the median trick to boost the confidence level (this can also be seen as a median of means trick).

Lecture 2

Distinct Element Counting

We turn to the problem of counting the number of distinct elements in a stream. Specifically, consider a stream of m numbers $x_1, \ldots, x_m$, where each $x_i \in [n]$. We want to construct an algorithm to output the number of distinct elements seen in the stream. There are several naive algorithms that should immediately come to mind. First, we can simply store a bit array of length n (requiring $O(n)$ bits of storage) and keep track of elements seen so far. Similarly, we can store all the input elements (requiring $O(m \log n)$ bits of storage). However, if m or n is very large, both these algorithms can be infeasible. We now turn to the Flajolet-Martin algorithm for distinct element counting, which uses roughly $O(\log n)$ bits of storage.

2.1 Flajolet-Martin Algorithm

The idea behind the Flajolet-Martin algorithm is to make use of a specific hash function oracle. Suppose that we were given access to a random hash function $h : [n] \to [0, 1]$. Specifically, for any number in 1 to n, the hash function would give a real number from 0 to 1, and such a hash function can be picked once at the beginning of the algorithm and queries to this function can be made afterwards. For now, we simply suppose that such a a hash function exists and we can efficiently store it. This assumption seems absurd at first glance since storing real numbers requires infinite storage, but we will prove later that this hash function can indeed be constructed efficiently. Given this hash function h, the Flajolet-Martin (FM) algorithm is summarized in Algorithm 2.4.

Algorithm 2.4 Flajolet-Martin (FM) Algorithm for Distinct Element Counting

```
Given: Random hash function $h : [n] \to [0, 1]$
  Initialize variable $Z = 1$.
  Upon seeing an element $X = i$, set $Z = \min(Z, h(i))$.
  Return estimator $E = \frac{1}{Z} - 1$.
```

The intuition behind this algorithm is the counter we store, Z, will store the smallest hashed value seen in the stream. Notice that for duplicate elements in the stream, the value of Z does not change because of the static hash function h. The minimum hashed value seen should, in expectation, store some information about the number of elements seen. This is because, in expectation, a random hash function will uniformly distribute each element in the domain over $[0, 1]$. This will be formalized in more detail below. We now analyze the FM algorithm. Let d be the true number of distinct elements in the stream.

Theorem 2.1.1. *The counter Z in by the Flajolet-Martin algorithm has the following property:*

$$\mathbb{E}[Z] = \frac{1}{d+1}$$

Proof. Observe that duplicate elements in the stream do not affect the value of Z. Thus, Z is the minimum of d random variables lying in the range $[0,1]$. Pick any $a \in [0,1]$ at random. The probability that a is less than Z is $\frac{1}{d+1}$, since this is the probability that a is the smallest of $d+1$ numbers distributed independently over the interval $[0,1]$ (the probability that the $d+1$ numbers are all equal is 0 by properties of a continuous density). Note that this probability is also exactly equal to $\mathbb{E}[Z]$, since $\mathbb{E}[Z]$ is the expected value of the minimum of d random variables in $[0,1]$. Equating these two quantities, we have that $\mathbb{E}[Z] = \frac{1}{d+1}$. □

Theorem 2.1.2.

$$\operatorname{Var}[Z] \leq \frac{2}{d^2}$$

Proof. We know that $\operatorname{Var}[Z] = \mathbb{E}[Z^2] - \mathbb{E}[Z]^2 \leq \mathbb{E}[Z^2]$. Thus, we focus on solving $\mathbb{E}[Z^2]$. From one of the identities of expectation (Lemma A.3.1 in the Appendix):

$$\mathbb{E}[Z] = \int_0^\infty \Pr[Z > a]\ da$$

We can apply a similar analysis to the random variable Z^2:

$$\begin{aligned}
\mathbb{E}[Z^2] &= \int_0^\infty \Pr[Z^2 > a]\ da \\
&= \int_0^1 \Pr[Z > \sqrt{a}]\ da && \text{(Since } Z \in [0,1]\text{)} \\
&= \int_0^1 \Pr\left[\forall i \in \text{stream}, h(i) > \sqrt{a}\right]\ da \\
&= \int_0^1 \prod_{i \in \text{stream}} \Pr[h(i) > \sqrt{a}]\ da \\
&= \int_0^1 (1-\sqrt{a})^d\ da
\end{aligned}$$

The above step follows from the observation that for any i, $\Pr[h(i) \leq \sqrt{a}] = \sqrt{a}$ because $\sqrt{a}$ is between 0 and 1 and $h(i)$ is distributed uniformly between 0 and 1. There are d such probabilities since Z^2 does not change for duplicate stream elements and thus, these elements does not affect the expected value. We can now solve for this integral using traditional u-substitution:

$$\begin{aligned}
u &= 1 - \sqrt{a} \\
du &= \frac{1}{2\sqrt{a}}\ da
\end{aligned}$$

Solving for $\sqrt{a}$ in the first line gives that $\sqrt{a} = 1 - u$. We can plug this in for $\sqrt{a}$ in the second line, and solving for da gives that $da = 2(1-u)du$. Thus, the above integral becomes:

$$\begin{aligned}
\int_0^1 (1-\sqrt{a})^d \, da &= 2\int_0^1 (1-u)u^d \, du \\
&= 2\int_0^1 u^d - u^{d+1} \, du \\
&= \frac{2}{d+1}u^{d+1}\Big|_0^1 - \frac{2}{d+2}u^{d+2}\Big|_0^1 \\
&= \frac{2}{d+1} - \frac{2}{d+2} \\
&= \frac{2}{(d+1)(d+2)} \\
&\le \frac{2}{d^2}
\end{aligned}$$

This proves the claim. □

Note that given the above two claims, if we immediately attempt to apply Chebyshev's inequality, we get that $\Pr\left[|Z-\mathbb{E}[Z]| > \epsilon\mathbb{E}[Z]\right] \le \frac{\mathrm{Var}[Y]}{\epsilon^2\mathbb{E}[Y]^2} = \frac{1}{\epsilon^2}$. For small ϵ, this inequality means nothing, so we have no guarantees on how Z concentrates around its mean. So, how can we get a $1+\epsilon$ approximation for Z in approximating $\frac{1}{d+1}$? There are two options.

First, we can run the FM algorithm k independent times to obtain k different counters $Z_1, \ldots, Z_k$. Then, we can output the estimator $E = \frac{1}{Z} - 1$, where $Z = \frac{1}{k}\sum_{i=1}^k Z_i$.

Theorem 2.1.3. *Let $k = O\left(\frac{1}{\epsilon^2}\right)$. Suppose we run the FM algorithm over k independent runs to obtain k counters, $Z_1, \ldots, Z_k$. Then, $Z = \frac{1}{k}\sum_{i=1}^k Z_i$, will be a $1+\epsilon$ approximation to $\frac{1}{d+1}$ with probability at least 90%.*

Proof. Similar to 1.2.1 and 1.2.2, we can show that $\mathbb{E}[Z] = \frac{1}{d+1}$ and $\mathrm{Var}[Z] \le \frac{2}{d^2k}$. Then, we apply Chebyshev's inequality with $\lambda^2 = 10\mathrm{Var}[Z]$. If $k = O\left(\frac{1}{\epsilon^2}\right)$, we have that Z will be in the range $\frac{1}{d+1} \pm \epsilon \cdot O\left(\frac{1}{d}\right)$ with probability at least 90%. □

Theorem 2.1.4. *A $(1+\epsilon)$-approximation to Z implies that we have a $(1+\epsilon)$ approximation for d using the estimator $E = \frac{1}{Z} - 1$.*

Proof. Assume we have a $(1+\epsilon)$-approximation to Z, which implies the following (for small $\epsilon \in [0, \frac{1}{2}]$):

$$\begin{aligned}
(1-\epsilon)\frac{1}{d+1} \le &Z \le (1+\epsilon)\frac{1}{d+1} \\
\frac{d+1}{1+\epsilon} \le &\frac{1}{Z} \le \frac{d+1}{1-\epsilon} \\
(1-2\epsilon)(d+1) \le &\frac{1}{Z} \le (1+2\epsilon)(d+1) \\
(1-2\epsilon)(d+1) - 1 \le &E \le (1+2\epsilon)(d+1) - 1 \\
(1-2\epsilon)d - 2\epsilon \le &E \le (1+2\epsilon)d + 2\epsilon \\
(1-4\epsilon)d \le &E \le (1+4\epsilon)d
\end{aligned}$$

Setting $\epsilon' = 4\epsilon$ since we don't particularly mind about constants, we have proven the claim. □

Observe that we can also use the median trick to boost the static confidence level to an arbitrary δ failure probability. This gives $k = O\left(\frac{1}{\epsilon}\log\frac{1}{\delta}\right)$. We will explore the second option of obtaining a $1+\epsilon$ approximation in the following section.

2.2 Bottom-k Algorithm

The bottom-k algorithm stores only 1 hash function instead of k different hash functions as Option 1 above. The algorithm pseudocode is described in Algorithm 2.5.

Algorithm 2.5 Bottom-k Algorithm for Distinct Element Counting

```
Given: Random hash function $h : [n] \to [0,1]$. Assume that $d$ is sufficiently large ($> k$).
  Initialize variables $Z_1 = Z_2 = \cdots = Z_k = 1$.
  Upon seeing an element $X = i$, maintain $Z_1 \le Z_2 \le \cdots \le Z_k$, i.e. the $k$ smallest hash function values seen so far in the stream.
  Return estimator $\hat{d} = \frac{k}{Z_k}$.
```

We now analyze the algorithm.

Theorem 2.2.1. *With $k = O\left(\frac{1}{\epsilon^2}\right)$, the probability that the estimator $\hat{d}$ is outside of the range $d \pm \epsilon d$ (i.e. $\hat{d}$ is a $1+\epsilon$ approximation to d) is at most* 0.1.

Proof. Without loss of generality, assume that the input stream is simply $1, \ldots, d$ in order (we can make this assumption since Z does not change with duplicate elements). First, we prove the claim that $\Pr\left[\hat{d} > d(1+\epsilon)\right] < 0.05$. Define d indicator variables $X_i = \mathbb{1}\left[h(i) < \frac{k}{(1+\epsilon)d}\right]$. Note that, because $\hat{d} = \frac{k}{Z_k}$, the given condition $\hat{d} > (1+\epsilon)d$ is equivalent to the condition that $Z_k < \frac{k}{(1+\epsilon)d}$. Because Z_k is the kth smallest hash value seen, we conclude that there must be at least k hash values i such that $h(i) < \frac{k}{(1+\epsilon)d}$. This implies that $\sum_{i=1}^{d} X_i \ge k$. We now calculate the expectation and variance of this sum:

$$\begin{aligned}
\mathbb{E}\left[\sum_{i=1}^{d} X_i\right] &= d \cdot \mathbb{E}\left[X_i\right] \\
&= \frac{k}{1+\epsilon} \\
\mathrm{Var}\left[\sum_{i=1}^{d} X_i\right] &= d\mathrm{Var}\left[X_i\right] \\
&\le d\mathbb{E}\left[X_i^2\right] \\
&= \frac{k}{1+\epsilon} \\
&\le k
\end{aligned}$$

We now apply a standard Chebyshev bound with $\lambda^2 = 10\mathrm{Var}\left[\sum_{i=1}^{d} X_i\right]$, which yields that

$$\Pr\left[\left|\sum_{i=1}^{d} X_i - \frac{k}{1+\epsilon}\right| > \sqrt{20k}\right] < 0.05$$

and consequently implies

$$\Pr\left[\sum_{i=1}^{d} X_i > \frac{k}{1+\epsilon} + \sqrt{20k}\right] < 0.05$$

Take the condition on the right. We will show it is less than or equal to k. Thus, because the sum of X_i is monotonically increasing, this would prove that the probability that $\sum_{i=1}^{d} X_i \geq k$ is at most 0.05. Taking $\epsilon < \frac{1}{2}$ and $k = O\left(\frac{1}{\epsilon^2}\right)$, we have the following:

$$\begin{aligned}\frac{k}{1+\epsilon} + \sqrt{20k} &\leq k(1-\epsilon+\epsilon^2) + \sqrt{20k} && \text{(Taylor Series Expansion)}\\ &\leq k - O\left(\frac{1}{\epsilon}\right)\\ &\leq k\end{aligned}$$

We can prove that $\Pr[\hat{d} < (1-\epsilon)d] < 0.05$ similarly by keeping X_i as defined and noting that there must be $d - k$ $h(i)$ such that $h(i) > \frac{k}{(1-\epsilon)d}$, implying that there are k $h(i)$ such that $h(i)$ with $h(i) \leq \frac{k}{(1-\epsilon)d}$. The rest of the analysis is identical to above. A union bound over these two events proves the claim. □

The above theorem proves that with $d = \Omega\left(\frac{1}{\epsilon^2}\right)$, we require $O\left(\frac{1}{\epsilon^2}\right)$ counters to be stored. We require d to be lower bounded as such because otherwise the estimator is a meaningless quantity since it will always be less than d.

We can now return to the question we left at the beginning of the Flajolet-Martin algorithm, which is: how can we construct such a hash function h and store it efficiently? The first issue is that the codomain of the hash function is the real line, which cannot be stored on a machine. There is a simple fix to this: discretize the codomain up to an additive $\frac{1}{M}$, i.e. transform $[0,1]$ into $\left\{0, \frac{1}{M}, \frac{2}{M}, \ldots, 1\right\}$. If $M \gg n^3$, then the probability that $d \leq n$ random numbers collide is at most $\frac{1}{n}$ (Lemma A.2.1 in the Appendix). Thus, the counter in the algorithm requires $O(\log M) = O(\log n)$ bits. Finally, we can also observe that any 2-wise independent hash function will work for the algorithm (see Definition A.2.1).

2.3 Space Lower Bounds

We can prove several lower bounds for storage for algorithms that solve the distinct elements problem.

Theorem 2.3.1. *Any deterministic and exact algorithm for the distinct elements problem must use $\Omega(n)$ bits of storage.*

Proof. We prove by contradiction. Suppose such an algorithm A exists that requires less than $\Omega(n)$ bits of storage. We will first introduce some notation. Consider some vector $y \in \{0,1\}^n$, and denote S_y as the stream corresponding to this vector. Specifically, S_y is a stream such that $x = i$ is in the stream if and only if $y_i = 1$ - thus, y is like an indicator vector for the elements present in the stream S_y. Formally, we can view A as a deterministic finite automata (DFA) state machine with a memory. Each state of this DFA corresponds to what is stored in the memory of A. Let $\sigma_{A,y}$ denote the memory of A after A is run on the stream S_y. Since the algorithm is deterministic, this memory must not change across iterations and moreover, A must return the answer given only the memory at the final state. Note that $|\sigma_{A,y}| \leq s$, where s is the total memory size of A (assumed to be less than $\Omega(n)$).

Now, we define a function $f : \{0,1\}^n \to \{0,1\}^s$, where $f(y) = \sigma_{A,y}$. We claim that f is injective. To see this, fix A after seeing the stream S_y and reaching the state $\sigma_{A,y}$. Now, feed the algorithm A another stream element $x = i$. If our estimate of the number of distinct elements goes up, then we know that $y_i = 0$ in the original stream since this is a new distinct element. Otherwise, we can infer that $y_i = 0$. Thus, given $\sigma_{A,y}$ (the memory contents which encode the number of distinct elements), we can recover y_i. This implies that f is one-to-one, or injective.

If f is injective, the size of the codomain must be at least the size of the domain, implying that $2^n \leq 2^s$. Taking the logarithm of both sides, we have that $s \geq n$, or the total memory size of the algorithm is $\Omega(n)$. Thus, we have reached a contradiction and the claim is proven. □

Theorem 2.3.2. *Any deterministic and approximate algorithm still needs $\Omega(n)$ bits of storage.*

Proof. We prove by contradiction. Suppose such an algorithm A exists. Without loss of generality, say A is deterministic and 1.1-approximate. The analysis below holds for any constant factor of approximation. $\square$

Lecture 3

Frequency Moments

Appendices

Appendix A

Probability

A.1 Concentration Inequalities

We use several important probability tail bounds throughout the lecture that are important for analysis. They are summarized below.

Theorem A.1.1 (Markov's inequality)**.** *For any non-negative random variable $X \geq 0$ and $\lambda > 0$, we have the following bound:*

$$\Pr\left[X \geq \lambda\right] \leq \frac{\mathbb{E}[X]}{\lambda}$$

or equivalently:

$$\Pr\left[X \geq \lambda \cdot \mathbb{E}[X]\right] \leq \frac{1}{\lambda}$$

Theorem A.1.2 (Chebyshev's inequality)**.** *Chebyshev's inequality states that for any random variable X and for all $\lambda > 0$, we have the following bound:*

$$\Pr\left[|X - \mathbb{E}\left[X\right]| \geq \lambda\right] \leq \frac{\mathrm{Var}[X]}{\lambda^2}$$

Often, we will apply Chebyshev's inequality with $\lambda^2 = 10\mathrm{Var}[X]$, which allows us to claim that X lies in $\mathbb{E}[E_n] \pm \lambda$ with probability at least 90%.

Theorem A.1.3 (Chernoff/Hoeffding bound)**.** *Suppose $X_1, \ldots, X_n$ are iid random variables that take values in $[0, 1]$. Let $\mu = \mathbb{E}\left[\sum_{i=1}^n X_i\right]$, and ϵ be a value in $[0, 1]$. Then, the Chernoff/Hoeffding bounds state:*

$$\Pr\left[\left|\sum_{i=1}^{n} X_i - \mu\right| > \epsilon\mu\right] \leq 2\exp\left\{-\frac{\epsilon^2\mu}{3}\right\}$$

These bounds give concentration inequalities on how far a sum of iid random variables will deviate from its expectation.

A.2 Hashing

Definition A.2.1 (k-wise independent hash function)**.** A family of hash functions $\mathcal{H} = \{h : U \to [m]\}$ is k-wise independent if for any k distinct keys $x_1, \dots, x_k \in U$ and any k hash codes $y_1, \dots, y_k \in [m]$, we have that:

$$\Pr_{h \in \mathcal{H}} [h(x_1) = y_1 \wedge \cdots \wedge h(x_k) = y_k] = \frac{1}{m^k}$$

This is equivalent to saying that for any fixed set of k distinct keys, if h is drawn randomly from $\mathcal{H}$, $h(x_1), \dots, h(x_k)$ are independent random variables.

Lemma A.2.1 (Birthday Paradox Hashing)**.** *Suppose we have a random hash function h that hashes numbers into the range $[d]$. Then, if we hash n numbers, the probability that at least one collision occurs is at least:*

$$\begin{aligned} &= 1 - \prod_{k=1}^{n-1} \left(1 - \frac{k}{d}\right) \\ &\approx 1 - e^{-\frac{n(n-1)}{2d}} \end{aligned}$$

As a corollary, if we have a hash function $h : [m] \to [d]$, if we pick $d \gg m^3$, then the probability of hashing $n \leq m$ numbers with no collisions is at at least $1 - \frac{1}{m}$.

Proof. The first claim should be evident from simple probability rules. We omit the proof for how the first statement is approximately equal to the second equation. We simply prove the corollary. From the above statement, the probability of no collisions is approximately $e^{-\frac{n(n-1)}{2d}}$. Manipulating this inequality, we have that:

$$\begin{aligned} e^{-\frac{n(n-1)}{2d}} &\geq e^{-\frac{m^2}{2d}} \\ &= e^{-\frac{1}{m}} && (d \gg m^3) \\ &\geq 1 - \frac{1}{m} && (e^x \geq 1 + x) \end{aligned}$$

□

A.3 Tricks

Lemma A.3.1. *For any non-negative random variable X, we can write the expectation of X as*

$$\mathbb{E}[X] = \int_0^\infty \Pr[X > y] \; dy$$

More generally, for any random variable X, we can write the expectation of X as

$$\mathbb{E}[Z] = \int_0^\infty \Pr[X > y] \; dy - \int_{-\infty}^0 \Pr[X < y] \; dy$$

Proof. We present a proof sketch for the first part of the claim that is not fully rigorous. For intuition, consider a discrete random variable that takes values in $\{1, 2, 3, 4\}$. Then, we can write the expected value as:

$$\begin{aligned}
\mathbb{E}[X] &= 1 \cdot \Pr[X = 1] + 2 \cdot \Pr[X = 2] + 3 \cdot \Pr[X = 3] + 4 \cdot \Pr[X = 4] \\
&= \Pr[X = 1] + \Pr[X = 2] + \Pr[X = 3] + \Pr[X = 4] \\
&\qquad + \Pr[X = 2] + \Pr[X = 3] + \Pr[X = 4] \\
&\qquad + \Pr[X = 3] + \Pr[X = 4] \\
&\qquad + \Pr[X = 4] \\
&= \Pr[X \geq 1] + \Pr[X \geq 2] + \Pr[X \geq 3] + \Pr[X \geq 4]
\end{aligned}$$

Similar intuition can apply to continuous random variables, although the technical proof is much more complex and requires series analysis. □

Appendix B

Miscellaneous

B.1 Coding Theory

Introduce the result about how a set $T_n \subset \{0,1\}^n$ exists with sufficiently large distance between elements and large element norms.