Probabilities, Vectors, and Matrices in CS	**CS 4850/5850**
Lecture Notes: Randomized Algorithms and Discrete Probability	
Instructor: Robert Kleinberg	**Jan–Feb 2025**

1 Balls and bins

We begin our study of randomized algorithms by analyzing one of the most elementary random sampling processes: drawing a sequence of independent, uniformly random elements of the set $[n] = \{1, 2, \ldots, n\}$. The colloquial expression "balls and bins" is often applied to such processes, because of the metaphor of a sequence of balls being thrown at random into n bins.

Balls and bins are used to model computing phenomena such as hashing in data structures and load balancing in distributed systems. (We'll say much more about hashing later in this course.) A key intuition is that the process tends to lead to a fairly even distribution of balls among the bins, since all of the bins are treated symmetrically. Some of the most basic questions about balls and bins concern the extent to which the loads of the different bins (i.e. the numbers of balls they each contain) deviate from a perfectly uniform distribution. In all of the following problems, assume m balls are thrown into n bins.

Birthday Paradox How likely is it that at least one bin contains more than one ball?

Coupon Collector Problem How likely is it that no bin is empty?

Load Balancing How likely is it that the maximum and minimum bin loads differ by a factor less than $1 + \varepsilon$?

The third question illustrates a notational convention that we will use throughout this course: unless otherwise specified, the notation ε represents a number in the range $0 < \varepsilon < 1$, whose value does not vary with the other parameters of the problem. (In this case, that means ε does not depend on the number of balls or bins.)

1.1 The Birthday Paradox

Let's make the unrealistic assumption that birthdays are uniformly distributed[1] over the 365 days of the calendar. How large must a group of people be, in order for the probability that at least two of them share the same birthday to exceed $\frac{1}{2}$? Counterintuitively, this happens as long as there are at least 23 people in the group. This is generally known as the "birthday paradox", though it would be more accurate to describe the phenomenon as *surprising* rather than *paradoxical*.

The birthday paradox is a thinly disguised question about balls and bins: people's birthdays are the balls, and dates on the calendar are the bins. Before delving into the analysis that formally

[1] In fact, the distribution of birthdays in the United States is quite far from uniform. There are almost twice as many people born on the most common birthday, September 9, than on the least common annually-occurring one, December 25. Of course, it is even less common for someone to have the birthday February 29 because that date only occurs once every four years.

justifies the birthday paradox, the following intuition is helpful. The probability that any two given people have the same birthday (under our simplifying assumption of uniformly-distributed birthdays) is $\frac{1}{365}$. In a group of 23 people the number of *pairs* of people is $\binom{23}{2} = 253$. By linearity of expectation, the expected number of pairs of people who share a birthday is $\frac{253}{365} = 0.693\ldots$. In light of this calculation, it become a bit less surprising that there is a significant probability that the group contains at least one pair of people who share a birthday.

Let's consider, more generally, the *collision probability* when throwing m balls into n bins — that is, the probability that at least one bin is occupied by more than one ball. It turns out that the best way to calculate the collision probability is to calculate the probability of *no collisions* and then subtract from 1. An exact formula for the probability of no collisions can easily be obtained by thinking about throwing the balls sequentially. When the first ball is thrown, there cannot be a collision. When the second ball is thrown, the probability that it doesn't collide with the first is $\frac{n-1}{n}$. More generally, if $k \leq n$ balls have been thrown and all of them occupy distinct bins, then there are $n-k$ remaining unoccupied bins, so the probability that the $(k+1)^{\text{th}}$ ball does not collide with any of its predecessors is $\frac{n-k}{n}$. Multiplying all of these probabilities together, we find that

$$\Pr(\text{no collision}) = \prod_{k=1}^{m-1} \frac{n-k}{n} = \prod_{k=1}^{m-1} \left(1 - \frac{k}{n}\right). \tag{1}$$

Expressions like the right side of equation (1) are painful to deal with because if you expand out the product using the distributive law, you get a sum of exponentially many terms. (In this case, 2^{m-1} terms.) To make progress, it's time to apply *the most useful inequality in the analysis of randomized algorithms.*

Fact 1.1. *The inequality*

$$1 + x \leq e^x$$

holds for all $x \in \mathbb{R}$, and the inequality is strict except when $x = 0$.

Proof. The proof of the inequality is an application of the Mean Value Theorem and case analysis. When $x = 0$ the two sides of the inequality are both equal to 1. When $x > 0$ the strict inequality asserts that $\frac{e^x - 1}{x} > 1$. This can be seen by applying the Mean Value Theorem to the function $f(x) = e^x$, to conclude that there exists some y in the interval $(0, x)$ such that

$$\frac{f(x) - f(0)}{x} = f'(y).$$

The left side equals $\frac{e^x - 1}{x}$, while the right side equals e^y, which is greater than 1 because $y > 0$. Similarly, when $x < 0$ the strict inequality asserts $\frac{e^x - 1}{x} < 1$. (Dividing by x reverses the direction of the inequality because x is negative.) This holds because, again by the Mean Value Theorem, $\frac{e^x - 1}{x}$ is equal to e^y for some y in the interval $(x, 0)$. As y is strictly negative, $e^y < 1$ and the inequality follows. □

Now, applying Fact 1.1 to the birthday paradox calculation, we find that

$$\Pr(\text{no collision}) < \prod_{k=1}^{m-1} e^{-k/n} = \exp\left(-\frac{1}{n}\sum_{k=1}^{m-1} k\right) = \exp\left(-\frac{m(m-1)}{2n}\right). \tag{2}$$

The right side becomes less than $\frac{1}{2}$ when $\frac{m(m-1)}{2n} > \ln(2)$, or equivalently when

$$\left(m - \tfrac{1}{2}\right)^2 > 2n\ln(2) + \tfrac{1}{4}.$$

Lemma 1.2. *If $m > \sqrt{2n\ln(2) + \frac{1}{4}} + \frac{1}{2}$ and m balls are thrown randomly into n bins then the collision probability is greater than $\frac{1}{2}$.*

When $n = 365$, the expression $\sqrt{2n\ln(2) + \frac{1}{4}} + \frac{1}{2}$ evaluates to 22.99994…, a shockingly close approximation to 23. For most applications of the birthday paradox, it is acceptable to overestimate the required value of m, using the function $\sqrt{2n}$ rather than the more cumbersome formula specified in the lemma.

You probably noticed that our analysis of the collision probability involved *overestimating* the probability of having no collisions, by approximating each factor $1 - \frac{k}{n}$ with the overestimate $e^{-k/n}$. With a little more work, we can quantify the amount of error due to this approximation and verify that the error is very small.

Fact 1.3. *If $0 < x < \frac{1}{2}$ then $e^{-x-x^2} < 1 - x$.*

Proof. By the Taylor series for the natural logarithm,

$$-\ln(1-x) = \sum_{k=1}^{\infty} \frac{x^k}{k} < x + \sum_{k=2}^{\infty} \frac{x^k}{2} = x + \frac{x^2}{2} \sum_{j=0}^{\infty} x^j = x + \frac{x^2}{2} \cdot \frac{1}{1-x} < x + x^2$$

where the last inequality follows from our assumption that $x < \frac{1}{2}$. The inequality $e^{-x-x^2} < 1 - x$ now follows by negating both sides and exponentiating. □

Applying this fact to the birthday paradox, we see that the probability of having no collisions can be bounded *from below* as follows.

$$\begin{aligned}\Pr(\text{no collision}) &> \prod_{k=1}^{m-1} \exp\left(-\frac{k}{n} - \frac{k^2}{n^2}\right) = \exp\left(-\sum_{k=1}^{m-1} \frac{k}{n}\right) \cdot \exp\left(-\sum_{k=1}^{m-1} \frac{k^2}{n^2}\right) \\ &= \exp\left(-\frac{m(m-1)}{2n}\right) \cdot \exp\left(-\frac{m(m-1)(2m-1)}{6n^2}\right).\end{aligned}$$

If we choose m such that $\frac{m(m-1)}{2n} \approx \ln(2)$ then

$$\frac{m(m-1)(2m-1)}{6n^2} \approx \frac{\ln(2)\cdot(2m-1)}{3n} \approx \frac{\ln(2)\cdot 2\cdot\sqrt{2n\ln(2)}}{3n} = \frac{(2\ln 2)^{3/2}\cdot\sqrt{n}}{3n} < \frac{1}{\sqrt{n}}$$

$$\exp\left(-\frac{m(m-1)(2m-1)}{6n^2}\right) > 1 - \frac{m(m-1)(2m-1)}{6n^2} > 1 - \frac{1}{\sqrt{n}}.$$

Hence, the formula $\exp\left(-\frac{m(m-1)}{2n}\right)$ overestimates the probability of no collision, but only by a factor less than $(1 - \frac{1}{\sqrt{n}})^{-1}$. With a little more effort, one can use this estimate to show that the minimum number of balls necessary to ensure $\Pr(\text{collision}) \geq \frac{1}{2}$ is very close to the formula given in Lemma 1.2; that formula overestimates the minimum number of balls by at most 2.

1.1.1 Application: Insecurity of cryptographic hash functions

A cryptographic hash function is a function h that takes a long string and compresses it to a "message digest" of some fixed length. For example, the SHA-1 hash function that was in use until around 2017 had a 160-bit output. The SHA-2 family that replaces it consists of six hash functions with output lengths ranging from 224 to 512 bits.

To be considered secure, a cryptographic hash function should be *collision-resistant*, meaning that it is computationally infeasible to find two distinct inputs x, y such that $h(x) = h(y)$. A simple method for attempting to "break" a collision-resistant hash function is the Birthday Attack, which consists of evaluating the hash function on uniformly-random inputs until two of them yield the same output. This is analogous to throwing balls into $n = 2^k$ bins, until there is a bin containing two balls. From Lemma 1.2 we know that the Birthday Attack is likely to succeed after roughly $\sqrt{2n} = 2^{(k+1)/2}$ attempts. However, cryptographic hash functions used in practice are not truly random functions, they are only conjectured to be computationally indistinguishable from random functions. Sometimes these conjectures are false: if the hash function has some structure that makes it distinguishable from a truly random function, it may be possible to exploit that structure to find a hash collision much more rapidly than the Birthday Attack. That is exactly what happened to SHA-1 in 2005, when Xiaoyun Wang (and two students) succeeded in finding a collision using about 2^{69} attempts, about 2000 times faster than the 2^{80} attempts predicted by the birthday paradox. This was later improved to 2^{63}, which is more than 100,000 times faster than the Birthday Attack. The success of these attacks prompted the transition from SHA-1 to SHA-2.

1.1.2 Application: Sample complexity of uniformity testing

Hypothesis testing is a common task in statistics: given m samples, and given a candidate distribution p, determine whether or not it is likely that the samples were drawn from p. Uniformity testing is the special case when p is the uniform distribution on the set $[n] = \{1, 2, \ldots, n\}$.

Here's one way to formalize the objective of uniformity testing. We would like to design an algorithm, parameterized by a pair of positive constants ε, δ, that takes a sequence of m samples drawn from some distribution on the set $[n]$, and it either outputs "uniform" or "not uniform." The algorithm will be considered *probably approximately correct (PAC)* if it satisfies the following two properties.

- If the samples are drawn from the uniform distribution, then with probability at least $1 - \delta$ the algorithm outputs "uniform".
- If the samples are drawn from a distribution q that is ε-far from uniform, in the sense that there is a subset $S \subseteq [n]$ with $q(S) > \frac{|S|}{n} + \varepsilon$, then with probability at least $1 - \delta$ the algorithm outputs "not uniform".

The *sample complexity of* (ε, δ)*-PAC uniformity testing* refers to the minimum $m = m(n)$ for which such an algorithm exists.

It is possible to use the birthday paradox to prove a simple lower bound on the sample complexity of uniformity testing. To see this, consider the problem of distinguishing between two data

generating processes.

1. Samples $x_1, \ldots, x_m$ are drawn from the uniform distribution on $[n]$.
2. A random subset $T \subset [n]$ of size $n/2$ is drawn. Then, samples $x_1, \ldots, x_m$ are drawn from the uniform distribution on T.

If q denotes the distribution from which the samples $x_1, \ldots, x_m$ are drawn, then q equals the uniform distribution in Case 1, whereas q is ε-far from the uniform distribution (for any $\varepsilon < \frac{1}{2}$) in Case 2. This is because in Case 2, $q(T) = 1$ whereas $|T|/n = \frac{1}{2}$.

If $m \leq \sqrt{n}$ then the uniformity tester faces an insurmountable dilemma: in both Case 1 and Case 2, its input sequence is probably just a random sequence of m distinct elements of $[n]$. Hence, it has no useful signal for distinguishing Case 1 from Case 2. However, probable approximate correctness requires distinguishing Case 1 from Case 2, since in one case q is uniform whereas in the other case it is ε-far from uniform.

Here's how to make this reasoning precise. Let $\mathcal{E}$ denote the event that the input sequence consists of m distinct elements of $[n]$. Let $\Pr_1(\mathcal{E})$ and $\Pr_2(\mathcal{E})$ denote the probabilities of event $\mathcal{E}$ when the process generating the sequence $x_1, \ldots, x_m$ is as described in Case 1 and Case 2, respectively. In both cases, since the data generating process is invariant under permutations of the set $[n]$, the output distribution conditional on event $\mathcal{E}$ must be the uniform distribution over the set $\Sigma_{n,m}$ of length-m sequences of distinct elements of $[n]$. Let σ denote the probability that the uniformity testing algorithm outputs "uniform" when given an input sequence drawn uniformly at random from $\Sigma_{n,m}$. Assuming the uniformity tester is (ε, δ)-PAC:

$$\begin{aligned}
\delta &\geq \Pr_1(\text{output "not uniform"}) \geq \Pr_1(\text{output "not uniform"} \,|\, \mathcal{E}) \cdot \Pr_1(\mathcal{E}) = (1-\sigma) \cdot \Pr_1(\mathcal{E}) \\
\delta &\geq \Pr_2(\text{output "uniform"}) \geq \Pr_2(\text{output "uniform"} \,|\, \mathcal{E}) \cdot \Pr_2(\mathcal{E}) = \sigma \cdot \Pr_2(\mathcal{E}) \\
2\delta &\geq (1-\sigma)\Pr_1(\mathcal{E}) + \sigma \Pr_2(\mathcal{E}) \geq \Pr_2(\mathcal{E}),
\end{aligned}$$

where the final inequality holds because the data generating process is less likely to generate m distinct samples in Case 2 than in Case 1.

Using our analysis of the birthday paradox, we know that

$$\Pr_2(\mathcal{E}) \geq \exp\left(-\frac{m(m-1)}{2|T|}\right) \cdot \left(1 - \frac{1}{\sqrt{|T|}}\right) = \exp\left(-\frac{m(m-1)}{n}\right) \cdot \left(1 - \sqrt{\tfrac{2}{n}}\right).$$

When $m < \sqrt{n}$ and $n \geq 8$, the right side is at least $\frac{1}{2e}$. Hence, (ε, δ)-PAC uniformity testing with $m < \sqrt{n}$ samples is not possible for any $\varepsilon < \frac{1}{2}$ and $\delta < \frac{1}{4e}$.

Is $m = \Theta(\sqrt{n})$ the correct sample complexity bound for uniformity testing? It turns out that the answer is yes. One might find this result surprising, because $O(\sqrt{n})$ samples constitute only a *tiny fraction of the support set of the distribution*, when n is large. It is amazing that such a small set of samples contains enough information to reliably distinguish uniform distributions from non-uniform ones. Not surprisingly, the the algorithm for uniformity testing using $O(\sqrt{n})$ samples relies heavily on the birthday paradox. In fact, it works by simply counting collisions: it outputs "uniform" if the number of pairs $i \neq j$ with $x_i = x_j$ falls below a carefully-designed threshold, and "not uniform" otherwise.

1.2 The Coupon Collector Problem

Continuing with our analysis of balls and bins, we turn to the question: how many balls must we throw to ensure that, with probability at least $\frac{1}{2}$, every bin contains at least one ball?

1.2.1 Reviewing the geometric distribution

To begin solving the coupon collector problem, we must recall the definition and some basic properties of the geometric distribution.

Definition 1.1. If X is a random variable taking values in the positive integers, with the probability distribution

$$\Pr(X = n) = p \cdot (1-p)^{n-1},$$

then we say X is *geometrically distributed with parameter p.*

If one takes a biased coin with $\Pr(\text{heads}) = p$ and tosses it until the first coin-toss that yields heads, the total number of tosses is geometrically distributed with parameter p.

Expectation and variance of a geometric random variable. It's not too hard to compute the expected value of a geometric random variable using the definition of the expectation:

$$\mathbb{E}[X] = \sum_{n=0}^{\infty} n \cdot \Pr(X = n).$$

However, it's even easier to use the following lemma, which frequently furnishes a very useful method for computing expected values or for bounding them from above or below.

Lemma 1.4. *If X is a random variable taking values in $\mathbb{N}$ then*

$$\mathbb{E}[X] = \sum_{k=0}^{\infty} \Pr(X > k).$$

Proof. Using the identity $n = \sum_{k=0}^{n-1} 1$, we find that

$$\mathbb{E}[X] = \sum_{n=0}^{\infty} n \cdot \Pr(X = n) = \sum_{n=0}^{\infty} \sum_{k=0}^{n-1} \Pr(X = n) = \sum_{k=0}^{\infty} \sum_{n=k+1}^{\infty} \Pr(X = n) = \sum_{k=0}^{\infty} \Pr(X > k).$$

□

For the geometric distribution, we have

$$\Pr(X > k) = \sum_{n=k+1}^{\infty} \Pr(X = n) = \sum_{n=k+1}^{\infty} p(1-p)^{n-1} = \sum_{n=k+1}^{\infty} (1-p)^{n-1} - (1-p)^{n} = (1-p)^{k}$$

since the sum telescopes. Hence,

$$\mathbb{E}[X] = \sum_{k=0}^{\infty} \Pr(X > k) = \sum_{k=0}^{\infty} (1-p)^k = \frac{1}{p}.$$

We can also use Lemma 1.4 to compute the variance of a geometric random variable. The key is to group the values of k into intervals between consecutive squares. If j is an integer and $j^2 \le k < (j+1)^2$ then $\Pr(X^2 > k) = \Pr(X > j) = (1-p)^j$. Consequently, the second moment of the geometric distribution satisfies

$$\begin{aligned}
\mathbb{E}[X^2] = \sum_{k=0}^{\infty} \Pr(X^2 > k) &= \sum_{j=0}^{\infty} \sum_{j^2 \le k < (j+1)^2} \Pr(X^2 > k) \\
&= \sum_{j=0}^{\infty} (2j+1)(1-p)^j \\
&= \sum_{i=1}^{\infty} (2i-1)(1-p)^{i-1} \\
&= \sum_{i=1}^{\infty} \frac{2i-1}{p} \cdot \Pr(X = i) = \mathbb{E}\left[\frac{2X-1}{p}\right] = \frac{2}{p^2} - \frac{1}{p}
\end{aligned}$$

where the last equation follows by linearity of expectation, substituting the formula $\mathbb{E}[X] = \frac{1}{p}$ that we already derived.

Recalling now that $\mathrm{Var}(X) = \mathbb{E}[X^2] - (\mathbb{E}[X])^2$, we find that

$$\mathrm{Var}(X) = \frac{1}{p^2} - \frac{1}{p} = \frac{1-p}{p^2}.$$

1.2.2 Coupon collector: distribution of stopping time

Consider throwing a sequence of balls into independent, uniformly random bins until the load in each bin is positive. Let τ denote the time when this stopping condition is met. The aim of this section is to compute the distribution of the random variable τ. To do so, it is useful to define random variables $\tau_1 < \tau_2 < \cdots < \tau_n = \tau$ by specifying that τ_k is the first time when there are k bins each containing at least one ball. Note that τ_1 is deterministically equal to 1. For the remaining random variables in this sequence, we have the following extremely useful observation.

Lemma 1.5. *For $1 \le k < n$, the random variable $Y_k = \tau_{k+1} - \tau_k$ is geometrically distributed with parameter $\frac{n-k}{n}$. These random variables $Y_1, Y_2, \ldots, Y_{n-1}$ are mutually independent.*

Proof. By definition, τ_{k+1} is the first time after τ_k that a ball is thrown into an unoccupied bin. For any t such that $\tau_k \le t < \tau_{k+1}$, the number of occupied bins at time t equals k. Hence, the probability that that ball thrown at time t lands in an unoccupied bin is $\frac{n-k}{n}$. It follows that the number of balls thrown after τ_k until one of them lands in an unoccupied bin — that is, the random variable $Y_k = \tau_{k+1} - \tau_k$ — follows a geometric distribution with parameter $\frac{n-k}{n}$. Since this distribution has no dependence on the history of the balls-and-bins preceding τ_k or following τ_{k+1}, we may conclude that $Y_1, \ldots, Y_{n-1}$ are mutually independent. □

If we define Y_0 to be a random variable that is deterministic equal to 1, then we have shown that the random stopping time $\tau = \tau_n$ can be represented as a sum of independent random variables $Y_0 + Y_1 + \cdots + Y_{n-1}$, where Y_k is geometrically distributed with parameter $\frac{n-k}{n}$. Consequently,

$$\mathbb{E}[\tau] = \sum_{k=0}^{n-1} \mathbb{E}[Y_k] = \sum_{k=0}^{n-1} \frac{n}{n-k} = n \cdot \left(1 + \frac{1}{2} + \cdots + \frac{1}{n}\right) = n \cdot H_n$$

The number of $H_n = 1 + \frac{1}{2} + \cdots + \frac{1}{n}$ is called the n^{th} *harmonic number* and lies between $\ln n$ and $1 + \ln n$, by the integral test:

$$\ln n = \int_1^n \frac{dx}{x} < 1 + \frac{1}{2} + \cdots + \frac{1}{n-1} < H_n = 1 + \frac{1}{2} + \cdots + \frac{1}{n} \leq 1 + \int_1^n \frac{dx}{x} = 1 + \ln n. \quad (3)$$

Finally, for the variance of τ, we have the following bound.

$$\mathrm{Var}(\tau) = \sum_{k=0}^{n-1} \mathrm{Var}(Y_k) = \sum_{k=0}^{n-1} \frac{k/n}{(n-k)^2/n^2} = n \sum_{k=1}^{n-1} \frac{k}{(n-k)^2} = n \sum_{j=1}^{n-1} \frac{n-j}{j^2} < n^2 \sum_{j=1}^{n-1} \frac{1}{j^2} < 2n^2.$$

The last inequality follows from a telescoping sum argument: $\sum_{j=1}^{n-1} \frac{1}{j^2} < \sum_{j=1}^{n-1} \left(\frac{2}{j} - \frac{2}{j+1}\right) = 2 - \frac{2}{n}$.

In the following section we'll see how to combine these estimates of the expectation and variance of τ to derive asymptotically tight bounds for the coupon collector problem.

1.2.3 Coupon collector: Tail bounds on stopping time

Let $m_{\text{coupon}}(n)$ denote the least value of m such that, when m balls are thrown into n bins, with probability at least $\frac{1}{2}$ every bin is occupied by at least one ball. We can relate $m_{\text{coupon}}(n)$ to the sequential balls-and-bins process analyzed in the preceding section: it is the least value of m such that $\Pr(\tau \leq m) \geq \frac{1}{2}$.

To find an upper bound for $m_{\text{coupon}}(n)$, the easiest approach is to use Markov's inequality, which asserts that for any non-negative random variable X and any factor $c \geq 1$,

$$\Pr(X \geq c \cdot \mathbb{E}X) \leq \frac{1}{c}.$$

Applying this inequality with $X = \tau$ and $c = 2$, we find that

$$\Pr(\tau \geq 2nH_n) \leq \frac{1}{2}$$

and consequently $m_{\text{coupon}}(n) \leq 2nH_n$.

Markov's inequality is quite a weak inequality, so applications of Markov's inequality rarely give tight or nearly-tight bounds on the quantity of interest. This case is no exception: the estimate $m_{\text{coupon}}(n) \leq 2nH_n \approx 2n \ln n$ is off by about a factor of 2.

To obtain a tighter bound, we use Chebyshev's inequality, which is simply Markov's inequality applied to the random variable $Z = (\tau - \mathbb{E}[\tau])^2$. By our calculation of $\mathrm{Var}(\tau)$, we know that $\mathbb{E}[Z] < 2n^2$. Hence,

$$\Pr\left(|\tau - \mathbb{E}[\tau]| \geq 2n\right) \;=\; \Pr(Z \geq 4n^2) \;\leq\; \Pr(Z \geq 2\mathbb{E}[Z]) \;\leq\; \frac{1}{2}. \tag{4}$$

A simple consequence of Inequality (4) is that

$$\mathbb{E}[\tau] - 2n \leq m_{\text{coupon}}(n) \leq \mathbb{E}[\tau] + 2n$$

which we can rewrite, using $\mathbb{E}[\tau] = nH_n$ and $\ln n < H_n \leq \ln n + 1$, as

$$n(\ln n - 2) \leq m_{\text{coupon}}(n) \leq n(\ln n + 3).$$

This improves, by approximately a factor of 2, our previous bound $m_{\text{coupon}}(n) \leq 2\mathbb{E}[\tau] = 2nH_n$. Importantly, the improved bound is asymptotically tight, i.e. the upper and lower bounds differ by a factor that converges to 1 as $n \to \infty$. To see why, note that $\frac{n(\ln n+3)}{n(\ln n-2)} = 1 + \frac{5}{\ln n-2}$.

1.3 Balls and bins: the heavily loaded case

We now turn to investigating the balls-and-bins problem from the standpoint of load balancing. Let us say the bins are β-balanced if the maximum bin load is no more than β times the minimum bin load. In order for the bins to be β-balanced, the minimum bin load must be strictly positive. In the analysis of the coupon collector problem, we saw that this happens for the first time at $\tau \approx n \ln n$. At the coupon-collector stopping time, the minimum bin load is equal to 1 (by definition of τ) and the maximum bin load is at least $\frac{\tau}{n}$, (by the pigeonhole principle), so the so bins are not β-balanced for any $\beta < \frac{\tau}{n}$, which is (probably) approximately $\ln n$.

As we throw even more balls into the bins, we expect the bins to become β-balanced for ever-smaller load factors β, with $\beta \to 1$ as the number of balls approaches ∞. Our objective in this section is to analyze how rapidly the load factor approaches 1. What is the smallest $m = m_\varepsilon(n)$ that ensures $\beta \leq 1 + \varepsilon$ with probability at least $\frac{1}{2}$?

1.3.1 The tail-bound-plus-union-bound method

Our strategy will exemplify a very commonly adopted method for analyzing randomized algorithms: the tail-bound-plus-union-bound method. The union bound is the following extremely simple yet useful probabilistic inequality.

Lemma 1.6. *If $\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_n$ is a finite collection of events in a probability space, then*

$$\Pr(\mathcal{E}_1 \cup \mathcal{E}_2 \cup \cdots \cup \mathcal{E}_n) \leq \Pr(\mathcal{E}_1) + \Pr(\mathcal{E}_2) + \cdots + \Pr(\mathcal{E}_n). \tag{5}$$

Armed with the union bound, we will execute the following strategy for analyzing the load balancing factor of the balls-and-bins process.

1. Let B_i denote the (random) number of balls in bin i, after m balls have been thrown.
2. Observe that $\mathbb{E}[B_i] = m/n$.
3. For any δ such that $0 < \delta < 1$, let $\mathcal{E}_{i,\delta}$ denote the event that $B_i \notin \left[(1-\delta)\frac{m}{n}, (1+\delta)\frac{m}{n}\right]$.
4. *(Tail bound step.)* Choose m large enough, as a function of n and δ, to ensure that
$$\forall i \quad \Pr(\mathcal{E}_{i,\delta}) \leq \tfrac{1}{2n}.$$
5. *(Union bound step.)* By the union bound, $\Pr\left(\bigcup_{i=1}^{n} \mathcal{E}_{i,\delta}\right) \leq \left(\frac{1}{2n}\right) \cdot n = \frac{1}{2}$.
6. The complementary event $\bigcap_{i=1}^{n} \overline{\mathcal{E}_{i,\delta}}$ has probablity at least $\frac{1}{2}$. In other words, with probability at least $\frac{1}{2}$ the load in each bin satisfies $(1-\delta)\frac{m}{n} \leq B_i \leq (1+\delta)\frac{m}{n}$.
7. Now, choose δ small enough that when B_i is between $(1-\delta)\frac{m}{n}$ and $(1+\delta)\frac{m}{n}$ for each i, it implies the bins are $(1+\varepsilon)$-balanced.

The last step is easy to accomplish: as long as $\varepsilon \leq 1$, we can set $\delta = \varepsilon/3$ and observe that

$$(1-\delta)(1+\varepsilon) = 1 + \frac{2\varepsilon}{3} - \frac{\varepsilon^2}{3} \geq 1 + \frac{\varepsilon}{3} = 1 + \delta,$$

which implies

$$\frac{1+\delta}{1-\delta} \leq 1 + \varepsilon.$$

To complete the strategy specified above we need to fill in the details of the tail bound step. First we'll see how to do this using Chebyshev's Inequality. Then we'll see a different tail bound, the Chernoff Bound, that is quantitatively much stronger, leading to a much tighter upper bound on $m_\varepsilon(n)$.

1.3.2 Tail bound using Chebyshev's inequality

Our strategy for bounding the probability that B_i lies outside the interval $\left[(1-\delta)\frac{m}{n}, (1+\delta)\frac{m}{n}\right]$ will be to express B_i as a sum of independent random variables. Specifically, let $X_{it} = 1$ if ball t is thrown into bin i, and $X_{it} = 0$ otherwise. The variables X_{it} are independent Bernoulli random variables, each with expected value $\frac{1}{n}$ and variance $\frac{1}{n}(1-\frac{1}{n})$. Their sum, B_i, has expected value $\frac{m}{n}$ and variance $\frac{m}{n}(1-\frac{1}{n})$, since the variance of a sum of independent random variables is the sum of their variances.

Now, using Chebyshev's inequality,

$$\Pr(\mathcal{E}_{i,\delta}) = \Pr\left(|B_i - \mathbb{E}[B_i]| > \frac{\delta m}{n}\right) \leq \frac{\mathrm{Var}(B_i)}{(\delta m/n)^2} < \frac{m/n}{(\delta m/n)^2} = \frac{n}{\delta^2 m}.$$

To make the right side less than or equal to $\frac{1}{2n}$ we may choose $m \geq 2(n/\delta)^2$. By our choice of $\delta = \varepsilon/3$, this yields the bound

$$m_\varepsilon(n) \leq \frac{18n^2}{\varepsilon^2}.$$

In the next two subsections, we'll see how to improve the dependence on n from quadratic to quasi-linear using a stronger tail bound.

1.3.3 Introducing the Chernoff Bound

Let $X_1, X_2, \ldots, X_m$ be independent (not necessarily identically distributed) random variables taking values in $[0, 1]$. In this section we derive the *Chernoff bound*, which bounds the probability that $X_1 + \cdots + X_m$ differs from its expectation by a factor lying outside the interval $[1 - \varepsilon, 1 + \varepsilon]$. We will assume throughout this section that $0 < \varepsilon < 1$.

The Chernoff bound is proven using the same strategy as Chebyshev's inequality.

1. Find a useful non-linear function of the random variable $X = X_1 + \cdots + X_m$.
   - In Chebyshev's inequality this function was $f(X) = (X - \mathbb{E}[X])^2$.
   - In the Chernoff bound it will be $f(x) = e^{tX}$ for a carefully-chosen value of t.
2. Calculate an upper bound on $\mathbb{E}[f(X)]$.
3. Show that when X is far from $\mathbb{E}[X]$, the value of $f(X)$ exceeds this upper bound by a large factor.
4. Finish up by using Markov's inequality to assert that $f(X)$ is unlikely to exceed $\mathbb{E}[f(X)]$ by a large factor.

Definition 1.2. If X is a random variable, its *moment generating function* $M_X(t)$ and its *cumulant generating function* $K_X(t)$ are the functions defined by

$$M_X(t) = \mathbb{E}[e^{tX}]$$
$$K_X(t) = \ln M_X(t).$$

The following two lemmas present some useful properties of $M_X(t)$ and $K_X(t)$.

Lemma 1.7. *If $X_1, \ldots, X_m$ are independent random variables then*

$$M_X(t) = \prod_{i=1}^{m} M_{X_i}(t), \qquad K_X(t) = \sum_{i=1}^{m} K_{X_i}(t).$$

Proof. The formula for $M_X(t)$ follows from the fact that $e^{tX} = \prod_{i=1}^{m} e^{tX_i}$ and that the expectation of the product of independent random variables equals the product of their expectations. *(This is why it's crucial, in the Chernoff bound, that the variables must be independent.)* The formula for $K_X(t)$ follows from the one for $M_X(t)$, using the product rule for logarithms. □

Remark 1.1. When $M_X(t)$ and $K_X(t)$ are expanded as power series in t,

$$M_X(t) = \sum_{n=0}^{\infty} \frac{m_n(X)}{n!} \cdot t^n, \qquad K_X(t) = \sum_{n=0}^{\infty} \frac{\kappa_n(X)}{n!} \cdot t^n,$$

the power series coefficients $m_n(X)$ and $\kappa_n(X)$ are the expected values of degree-n polynomials in X. Both of these sequences of coefficients represent important statistics about the distribution of X.

- $m_n(X) = \mathbb{E}[X^n]$ is the n^{th} moment of X.
- $\kappa_n(X)$ is called the n^{th} *cumulant* of X. The first two cumulants are well known statistics: $\kappa_1(X)$ is the expectation and $\kappa_2(X)$ is the variance. All of the cumulants satisfy the *cumulative property for independent sums:*

$$\kappa_n(X) = \kappa_n(X_1) + \kappa_n(X_2) + \cdots + \kappa_n(X_m)$$

when $X = X_1 + \cdots + X_m$ is a sum of independent random variables. This follows from the fact that $K_X(t) = \sum_{i=1}^m K_{X_i}(t)$.

Lemma 1.8. *For any random variable X taking values in $[0,1]$, the moment generating function M_X satisfies*

$$M_X(t) \le \exp\left((e^t - 1)\mathbb{E}[X]\right)$$

for all $t \in \mathbb{R}$.

Proof. For all $x \in [0,1]$ and all $t \in \mathbb{R}$ the inequality

$$e^{tx} \le 1 + (e^t - 1)x$$

holds because the left side is a convex function of x, the right side is a linear function of x, and the left and right sides are equal at the endpoints $x = 0$ and $x = 1$. Applying this inequality along with linearity of expectation, we find that

$$M_X(t) = \mathbb{E}\left[e^{tX}\right] \le 1 + (e^t - 1)\mathbb{E}[X].$$

The lemma follows by combining this inequality with Fact 1.1. □

Theorem 1.9 (Chernoff bound)**.** *If $X_1, X_2, \ldots, X_m$ are independent random variables taking values in $[0,1]$ and $X = X_1 + \cdots + X_m$, then for $0 < \varepsilon \le 1$ we have*

$$\begin{aligned}\Pr(X \ge (1+\varepsilon)\mathbb{E}[X]) &< e^{-\frac{1}{3}\varepsilon^2\mathbb{E}[X]} \\ \Pr(X \le (1-\varepsilon)\mathbb{E}[X]) &< e^{-\frac{1}{2}\varepsilon^2\mathbb{E}[X]}\end{aligned}$$

Proof. Using Lemmas 1.7 and 1.8, together with $\mathbb{E}[X] = \sum_{i=1}^m \mathbb{E}[X_i]$, we find that $M_X(t) \le \exp\left((e^t - 1)\mathbb{E}[X]\right)$ for all $t \in \mathbb{R}$. Now, from Markov's inequality we have

$$\Pr(X \ge (1+\varepsilon)\mathbb{E}[X]) = \Pr\left(e^{tX} \ge e^{(1+\varepsilon)t\mathbb{E}[X]}\right) < e^{(e^t - 1 - (1+\varepsilon)t)\mathbb{E}[X]}$$

for all $t \ge 0$. To minimize the right side, set $t = \ln(1+\varepsilon)$. Then $e^t - 1 - (1+\varepsilon)t = \varepsilon - (1+\varepsilon)\ln(1+\varepsilon)$. Using the Taylor series

$$(1+\varepsilon)\ln(1+\varepsilon) = (1+\varepsilon)\left(\varepsilon - \tfrac{1}{2}\varepsilon^2 + \tfrac{1}{3}\varepsilon^3 - \cdots\right) = \varepsilon + \tfrac{1}{2}\varepsilon^2 - \tfrac{1}{6}\varepsilon^3 + \cdots > \varepsilon + \tfrac{1}{3}\varepsilon^2$$

we find that $\varepsilon - (1+\varepsilon)\ln(1+\varepsilon) < -\frac{1}{3}\varepsilon^2$ and the upper bound on $\Pr(X \ge (1+\varepsilon)\mathbb{E}[X])$ follows.

For $t \ge 0$ another application of Markov's inequality yields

$$\Pr(X \le (1-\varepsilon)\mathbb{E}[X]) = \Pr\left(e^{-tX} \ge e^{-(1-\varepsilon)t\mathbb{E}[X]}\right) < e^{(e^{-t} - 1 + (1-\varepsilon)t)\mathbb{E}[X]}.$$

To minimize the right side we set $t = -\ln(1-\varepsilon)$ and then $e^{-t} - 1 + (1-\varepsilon)t = -\varepsilon - (1-\varepsilon)\ln(1-\varepsilon)$. Using the Taylor series

$$-(1-\varepsilon)\ln(1-\varepsilon) = (1-\varepsilon)(\varepsilon + \tfrac{1}{2}\varepsilon^2 + \tfrac{1}{3}\varepsilon^3 + \cdots) = \varepsilon - \tfrac{1}{2}\varepsilon^2 - \tfrac{1}{6}\varepsilon^3 - \cdots < \varepsilon - \tfrac{1}{2}\varepsilon^2$$

we find that $-\varepsilon - (1-\varepsilon)\ln(1-\varepsilon) < -\frac{1}{2}\varepsilon^2$ and the upper bound on $\Pr(X \le (1+\varepsilon)\mathbb{E}[X])$ follows. □

A few features of the Chernoff bound are worth noting.

1. Theorem 1.9 bounds the probability of X deviating from $\mathbb{E}[X]$ by a large amount. Inequalities of this type are called *large deviation inequalities* or *tail bounds*, since they quantify the amount of probability in the "tail" of the distribution of X.

2. The probability of a large deviation tends to zero *exponentially fast* as $\mathbb{E}[X]$ grows large. Inequalities of this type are called *exponential tail bounds*.

3. The probability of large deviation is exponentially small as a function of $\mathbb{E}[X]$, not as a function of the number of random variables being summed, m. Even if m is very large, it's possible that the distribution of X is not very concentrated around its expected value. For example, if $X_1, \ldots, X_{m-1}$ are deterministically equal to 0, and X_m is equal to 0 or 1, each with probability $\frac{1}{2}$, then X is equal to 0 or 1, each with probability $\frac{1}{2}$, so the event that X is between $(1-\varepsilon)\mathbb{E}[X]$ and $(1+\varepsilon)\mathbb{E}[X]$ has probability zero! This is consistent with the Chernoff bound, which only says that $\Pr(X \ge (1+\varepsilon)\mathbb{E}[X])$ is small when $\mathbb{E}[X]$ is large.

4. In the exponential function on the right side of the Chernoff bound, the dependence on ε is quadratic. This is typical of exponential tail bounds. In order for a deviation such as $X \ge (1+\varepsilon)\mathbb{E}[X]$ to be unlikely, the expected value of X must be greater than $1/\varepsilon^2$ times the maximum value of any individual X_i. A useful way of summarizing this observation is, "*To estimate the frequency of an event within a factor of* $1 \pm \varepsilon$*, you must wait until you have observed the event at least* $1/\varepsilon^2$ *times.*"

1.3.4 Using the Chernoff Bound to analyze balls and bins

Let us now return to analyzing the load factor in the balls-and-bins process: the ratio of the maximum and minimum bin loads after m balls have been thrown into n bins. We previously saw that, with probability at least $\frac{1}{2}$, the load factor is less than $1+\varepsilon$ once $m \ge \frac{18n^2}{\varepsilon^2}$ balls have been thrown. We will now show that this bound can be significantly improved using the Chernoff bound.

Recall the tail-bound-plus-union-bound strategy from Sections 1.3.1 and 1.3.2. The load in bin i after m balls have been thrown is a random variable $B_i = X_{i1} + \cdots + X_{im}$ where $X_{i1}, \ldots, X_{im}$ are independent Bernoulli random variables. If m is large enough that the event has probability less than $\frac{1}{2n}$, then the union bound implies that with probability at least $\frac{1}{2}$, the ratio of the maximum and minimum bin loads is less than $1+\varepsilon$.

The Chernoff bound constrains the probability that $B_i > \left(1+\frac{\varepsilon}{3}\right)\frac{m}{n}$ or $B_i < \left(1-\frac{\varepsilon}{3}\right)\frac{m}{n}$. The first of these probabilities is bounded above by $\exp\left(-\frac{\varepsilon^2}{27}\cdot\frac{m}{n}\right)$ while the second is bounded above by

$\exp\left(-\frac{\varepsilon^2}{18} \cdot \frac{m}{n}\right)$. Hence, the sum of the two probabilities is less than $2\exp\left(-\frac{\varepsilon^2}{27} \cdot \frac{m}{n}\right)$. We seek a value of m that satisfies the inequality

$$2\exp\left(-\frac{\varepsilon^2}{27} \cdot \frac{m}{n}\right) \leq \frac{1}{2n}.$$

After taking the logarithm of both sides and doing some algebra, we discover that this is equivalent to

$$m > \frac{27n\ln(4n)}{\varepsilon^2}.$$

Compared to the bound of $\frac{18n^2}{\varepsilon^2}$ that we derived using Chebyshev's inequality, the improvement here is that the dependence on n is quasi-linear, i.e. $O(n\log n)$, rather than quadratic. In fact, the quasi-linear dependence on n is the best possible, because we know from our analysis of the coupon collector problem that when $m = o(n\log n)$, it is unlikely that every bin is occupied.

1.3.5 The Hoeffding Bound

In this section we derive a different exponential tail bound in which we once again have independent random variables $X_1, \ldots, X_n$, each taking values in a bounded interval, and their sum is denoted by X. This time, rather than proving that the ratio $X/\mathbb{E}[X]$ is unlikely to be far from 1, we wish to prove that the absolute difference $|X - \mathbb{E}[X]|$ is unlikely to be far from 0. In other words, whereas the Chernoff bound provides conditions under which $\mathbb{E}[X]$ is likely to be a good multiplicative approximation to X, we wish to understand conditions under which $\mathbb{E}[X]$ is likely to be a good additive approximation to X. The Hoeffding bound answers this question.

As before, the exponential tail bound will be proven by using generating functions to transform the stated inequality into an application of Markov's inequality. This time, it will be more convenient to work with the cumulant generating function rather than the moment generating function. The following lemma summarizes the implications of Markov's inequality when working with cumulant generating functions.

Lemma 1.10. *Let X be a random variable with cumulant generating function $K_X(t)$, and suppose $\lambda > 0$. For any $t > 0$,*

$$\Pr(X \geq \mathbb{E}[X] + \lambda) \leq e^{K_X(t) - t(\mathbb{E}[X]+\lambda)}$$
$$\Pr(X \leq \mathbb{E}[X] - \lambda) \leq e^{K_X(-t) + t(\mathbb{E}[X]-\lambda)}.$$

Proof. To derive the bound on $\Pr(X \geq \mathbb{E}[X] + \lambda)$, observe that the inequality $X \geq \mathbb{E}[X] + \lambda$ holds if and only if $e^{tX} \geq e^{t(\mathbb{E}[X]+\lambda)}$ and apply Markov's inequality. To derive the bound on $\Pr(X \leq \mathbb{E}[X] - \lambda)$, observe that the inequality $X \leq \mathbb{E}[X] - \lambda$ holds if and only if $e^{-tX} \geq e^{-t(\mathbb{E}[X]-\lambda)}$ and again apply Markov's inequality. □

The key new ingredient in the proof of Hoeffding's Inequality is the following lemma that furnishes an upper bound on the cumulant generating function of a random variable.

Lemma 1.11 (Hoeffding's Lemma)**.** *If X is a random variable supported on an interval $[a, b]$, with expected value μ, then the cumulant generating function $K_X(t)$ satisfies*

$$K_X(t) - \mu t \leq \frac{(b-a)^2 t^2}{8}.$$

Proof. The left side is the cumulant generating function of the random variable $X - \mu$, which has expected value zero, so we may replace X with $X - \mu$ if necessary and assume henceforth, without loss of generality, that $\mathbb{E}[X] = 0$. The lemma then asserts the inequality $K_X(t) \leq \frac{1}{8}(b-a)^2 t^2$. To prove this inequality, we will use Taylor's Theorem. We know $K_X(0) = 0$ from the definition of the cumulant generating function, and we know $K_X'(0) = 0$ since the derivative of K_X at 0 is the expectation of X. Hence, by Taylor's Theorem with Lagrange's remainder term, $K_X(t) = \frac{1}{2}K_X''(u)t^2$ for some u.

To conclude the proof, we need to prove that $K_X''(u) \leq \frac{1}{4}(b-a)^2$ for all u, when X is a random variable supported on $[a, b]$. We will prove this bound by constructing a new random variable Y supported on $[a, b]$ whose cumulant generating function $K_Y(t)$ satisfies

$$K_Y(t) = K_X(u + t) - K_X(u)$$

for all t. Then, taking the second derivative of both sides with respect to t, we will obtain $K_Y''(0) = K_X''(u)$. Recalling that $K_Y''(0)$ is equal to the variance of Y, we will be left with showing that the variance of any random variable supported on $[a, b]$ is less than or equal to $\frac{1}{4}(b-a)^2$. It will turn out that this inequality is quite easy to prove.

Let Y be a random variable obtained from X by "reweighting the probability of each support point z by the factor e^{uz}." If X has probability density function $f_X(z)$ this means that Y has probability density function $f_Y(z) = \frac{1}{Z}e^{uz}f_X(z)$, where the normalization factor $Z = \int_{-\infty}^{\infty} e^{uz}f_X(z)$ is chosen so that the equation $\int_{-\infty}^{\infty} f_Y(z)\,dz = 1$ holds, as required for a probability density function. More generally, i.e. whether or not X has a probability density function, as long as $\mathbb{E}[e^{uX}] < \infty$ we can define Y by specifying its cumulative distribution function:

$$\Pr(Y \leq y) = \frac{\mathbb{E}[e^{uX} \cdot \mathbf{1}_{X \leq y}]}{\mathbb{E}[e^{uX}]}.$$

Then, we can compute $M_Y(t)$ for a given t by making use of the following identity which is valid for any non-negative random variable Z and whose proof is basically the same as the proof of Lemma 1.4, substituting integrals in place of sums.

$$\mathbb{E}[Z] = \int_0^{\infty} \Pr(Z > z)\,dz. \tag{6}$$

Setting $Z = e^{tY}$ and using the substitution $z = e^{tw}$, we find that

$$\begin{aligned}
M_Y(t) = \mathbb{E}[e^{tY}] = \mathbb{E}[Z] &= \int_0^\infty \Pr(Z > z)\, dz \\
&= \int_{-\infty}^\infty \Pr(e^{tY} > e^{tw}) \cdot te^{tw}\, dw \\
&= \int_{-\infty}^\infty \Pr(Y > w) \cdot te^{tw}\, dw \\
&= \int_{-\infty}^\infty \frac{\mathbb{E}[e^{uX} \cdot \mathbf{1}_{X>w}]}{\mathbb{E}[e^{uX}]} \cdot te^{tw}\, dw \\
&= \frac{1}{M_X(u)} \int_{-\infty}^\infty \mathbb{E}[e^{uX} \cdot \mathbf{1}_{X>w}] \cdot te^{tw}\, dw \\
&= \frac{1}{M_X(u)} \mathbb{E}_X\left[e^{uX} \int_{-\infty}^X te^{tw}\, dw\right] = \frac{1}{M_X(u)} \mathbb{E}_X\left[e^{uX} e^{tX}\right] = \frac{M_X(u+t)}{M_X(u)}
\end{aligned}$$

The identity $K_Y(y) = K_X(u+t) - K_X(u)$ follows by taking the logarithm of both sides.

As observed earlier, to conclude the proof of the lemma we need only show that a random variable Y supported on the interval $[a, b]$ has variance at most $\frac{1}{4}(b-a)^2$. The validity of the inequality $\mathrm{Var}(Y) \le \frac{1}{4}(b-a)^2$ is unaffected if we apply an affine transformation to Y and we apply the same affine transformation to the interval $[a, b]$. In other words, if we replace Y with $cY + d$ and we replace $[a, b]$ with $[ca+d, cb+d]$, the validity of the inequality is unaffected because the variance of Y is scaled by c^2, and the squared-length of the interval is also scaled by c^2. Hence, without loss of generality (applying an affine transformation to Y and to $[a, b]$ if necessary) we can assume $[a, b] = [-1, 1]$ and $\frac{1}{4}(b-a)^2 = 1$. The variance of Y is $\mathbb{E}[Y^2] - \mathbb{E}[Y]^2$. The first term on the right side is clearly no greater than 1 because Y is supported on $[-1, 1]$. Since $\mathbb{E}[Y]^2 \ge 0$, it follows that $\mathbb{E}[Y^2] - \mathbb{E}[Y]^2 \le 1$, as desired. □

Theorem 1.12. *(Hoeffding's Inequality) Suppose $X_1, X_2, \ldots, X_n$ are independent random variables and that for each i, the support of X_i is contained in a bounded interval $[a_i, b_i]$. Let $X = X_1 + \cdots + X_n$. For any $\lambda > 0$,*

$$\Pr(X \ge \mathbb{E}[X] + \lambda) \le \exp\left(-\frac{2\lambda^2}{\sum_{i=1}^n (b_i - a_i)^2}\right)$$
$$\Pr(X \le \mathbb{E}[X] - \lambda) \le \exp\left(-\frac{2\lambda^2}{\sum_{i=1}^n (b_i - a_i)^2}\right).$$

Proof. Let $\mu_i = \mathbb{E}[X_i]$ for each i, and let $\mu = \sum_{i=1}^n \mu_i = \mathbb{E}[X]$. By Hoeffding's Lemma, $K_{X_i}(t) - \mu_i t \le \frac{1}{8}(b_i - a_i)^2 t^2$ for all t and all i. Summing over i,

$$K_X(t) - \mu t \le \frac{1}{8} \sum_{i=1}^n (b_i - a_i)^2 t^2.$$

Let $c = \frac{1}{8} \sum_{i=1}^n (b_i - a_i)^2$. By Lemma 1.10,

$$\Pr(X \ge \mathbb{E}[X] + \lambda) \le e^{K_X(t) - \mu t - \lambda t} \le e^{ct^2 - \lambda t}.$$

The proof concludes by setting $t = \lambda/(2c)$, so that $ct^2 - \lambda t = -\frac{\lambda^2}{4c} = -\frac{2\lambda^2}{\sum_{i=1}^n (b_i - a_i)^2}$. □

1.4 Further applications of the Chernoff and Hoeffding bounds

The Chernoff and Hoeffding bounds are some of the most versatile tools in the analysis of randomized algorithms and the average-case analysis of algorithms. In this section we will present a number of applications of both.

1.4.1 Estimating the expected value of a distribution

Suppose Y is a random variable taking values in an interval $[0, M]$ whose expected value we wish to estimate. Let $Y_1, Y_2, \ldots$ be a sequence of independent random variables, each having the same distribution as Y. One way to estimate $\mathbb{E}[Y]$ is to simply take the unweighted average of the first N samples,

$$\hat{Y} = \frac{1}{N}(Y_1 + \cdots + Y_N).$$

We wish to determine a value of N such that the error of the estimate is very unlikely to exceed ε:

$$\Pr(|\hat{Y} - \mathbb{E}[Y]| > \varepsilon) < \delta.$$

This type of guarantee is summarized by saying that the estimator $\hat{Y}$ is "probability approximately correct," often abbreviated as PAC.

By Hoeffding's Inequality,

$$\Pr(|\hat{Y} - \mathbb{E}[Y]| > \varepsilon) = \Pr(|Y_1 + \ldots + Y_N - N\mathbb{E}[Y]| > N\varepsilon) \leq 2\exp\left(-\frac{2N^2\varepsilon^2}{NM^2}\right) = 2\exp\left(-\frac{2N\varepsilon^2}{M^2}\right).$$

To make this less than δ, we require

$$\begin{aligned}
\exp\left(-\frac{2N\varepsilon^2}{M^2}\right) &< \frac{\delta}{2} \\
\exp\left(\frac{2N\varepsilon^2}{M^2}\right) &> \frac{2}{\delta} \\
\frac{2N\varepsilon^2}{M^2} &> \ln\left(\frac{2}{\delta}\right) \\
N &> \frac{M^2}{2\varepsilon^2}\ln\left(\frac{2}{\delta}\right).
\end{aligned}$$

This sample complexity bound has several features that are typical for estimation procedures that use independent, identically distributed samples to estimate a scalar quantity. The number of samples required depends inverse-quadratically on the tolerable level of "relative error;" in this example the tolerable relative error is ε/M because we are trying to estimate a quantity belonging to an interval of length M, and we tolerate additive error up to ε. On the other hand, the number of samples depends on logarithmically on the inverse of the "confidence parameter," δ, which governs the maximum failure probability that is deemed tolerable.

1.4.2 Generalization error of empirical risk minimization

We now show how the Hoeffding bound can be applied to the important problem of *generalization error* in machine learning. To keep the analysis as simple as possible, we will focus on the task of *hypothesis selection*, where there is a finite set of hypotheses and the learner aims to use a set of training data to choose a hypothesis that generalizes to unseen data.

We can model the hypothesis selection problem as follows. We have:

- a random variable Z taking values in a set $\mathcal{Z}$;
- a finite set of hypotheses, $\mathcal{H} = \{h_1, \ldots, h_m\}$;
- independent random variables $Z_1, Z_2, \ldots, Z_N$, each identically distributed to Z, collectively called the *training set*.
- a loss function $L : \mathcal{H} \times \mathcal{Z} \to [0, 1]$. The value $L(h, z)$ indicates how poorly hypothesis h fits data point z.

We assume that the learner is given the training set $\{Z_1, \ldots, Z_N\}$ but does not know the distribution from which the Z_i's were sampled. There are two important ways of evaluating a hypothesis h.

1. The *population loss* is $\overline{L}(h) = \mathbb{E}[L(h, Z)]$. This measures how well hypothesis h performs on *the actual distribution* from which the data is sampled, including data points that were not present in the training data.
2. The *empirical loss* is $\frac{1}{N}\sum_{i=1}^{N} L(h, Z_i)$. This has the advantage that it can be computed from the training data, unlike the population loss which can only be computed if one knows the data distribution.

Empirical risk minimization is the algorithm that selects the hypothesis h_{ERM} that minimizes empirical loss on the training set. The hope is that if the training set is a representative sample of the data distribution, then h_{ERM} will also perform near-optimally when evaluated according to population loss, even though it was selected to minimize empirical loss rather than population loss.

Theorem 1.13. *Let h_* denote the element of $\mathcal{H}$ that minimizes population loss. For any $0 < \varepsilon, \delta < 1$, if the number of data samples, N, satisfies $N > \frac{2}{\varepsilon^2}\ln(2m/\delta)$ then with probability at least $1 - \delta$, $\overline{L}(h_{ERM}) \leq \varepsilon + \overline{L}(h_*)$.*

Proof. Let $\phi(Z_1, \ldots, Z_N)$ denote the Boolean predicate:

$$\forall h \in \mathcal{H} \quad \left| \overline{L}(h) - \frac{1}{N}\sum_{i=1} L(h, Z_i) \right| < \frac{\varepsilon}{2}.$$

When $Z_1, \ldots, Z_n$ satisfy property ϕ, it implies that $\overline{L}(h_{ERM}) \leq \varepsilon + \overline{L}(h_*)$ because

$$\overline{L}(h_{ERM}) \leq \frac{\varepsilon}{2} + \frac{1}{N}\sum_{i=1} L(h_{ERM}, Z_i) \leq \frac{\varepsilon}{2} + \frac{1}{N}\sum_{i=1} L(h_*, Z_i) \leq \varepsilon + \overline{L}(h_*).$$

The first and third inequalities are applications of property ϕ, the second inequality follows from the definition of h_{ERM}.

To complete the proof we just need to show that $\Pr(\phi(Z_1, \ldots, Z_n)) \geq 1 - \delta$. For $j = 1, 2, \ldots, m$ let $\psi_j(Z_1, \ldots, Z_n)$ denote the predicate $\left|\overline{L}(h_j) - \frac{1}{N}\sum_{i=1}^{N} L(h_j, Z_i)\right| \geq \frac{\varepsilon}{2}$. The random variables $X_i = \frac{1}{N}L(h_j, Z_i)$ take values in $[0, \frac{1}{N}]$ and the expectation of their sum is $\overline{L}(h_j)$, so applying Hoeffding's inequality with $\lambda = \varepsilon/2$ yields

$$\Pr(\psi_j(Z_1, \ldots, Z_n)) \leq 2\exp\left(-\frac{\varepsilon^2 N}{2}\right) \leq \frac{\delta}{m},$$

by our assumption that $N > \frac{2}{\varepsilon^2}\ln(2m/\delta)$. The Union Bound (Lemma 1.6) implies that

$$\Pr(\vee_{j=1}^{m}\psi_j(Z_1, \ldots, Z_n)) \leq \sum_{j=1}^{m} \Pr(\psi_j(Z_1, \ldots, Z_n)) \leq \delta.$$

Since $\vee_{j=1}^{m}\psi_j(Z_1, \ldots, Z_n)$ is the negation of $\phi(Z_1, \ldots, Z_n)$, it follows that $\Pr(\phi(Z_1, \ldots, Z_n)) \geq 1 - \delta$ as claimed. □

1.5 Reducing error rate of randomized algorithms

Our last application of the Chernoff bound comes from the theory of randomized algorithms for decision problems. A decision problem is a problem whose output is an element of $\{0, 1\}$, with 0 representing "no" and 1 representing "yes." A decision problem belongs to the complexity class P if there is a deterministic polynomial-time algorithm — i.e., an algorithm running in time $O(n^c)$ where n is the input size (in bits) and c is a constant — that answers the decision problem correctly on every possible input. The complexity class BPP consists of decision problems Π having a randomized polynomial-time algorithm A that satisfies the following guarantee, where x denotes the problem input and r denotes the random string used by A.

$$\forall x \quad \Pr(A(x, r) \neq \Pi(x)) \leq \frac{1}{3}. \tag{7}$$

The random string r is assumed to be a uniformly random binary string whose length, $L(n)$, is bounded by a polnomial function of the input size, n. Property (7) is often stated equivalently as follows: if $\Pi(x) = 1$ then $\Pr(A(x, r) = 1) \geq \frac{2}{3}$, while if $\Pi(x) = 0$ then $\Pr(A(x, r) = 1) \leq \frac{1}{3}$.

The error rate of a BPP algorithm can be reduced by running it repeatedly using independent random strings, and taking a majority vote of the outcomes. The following algorithm uses a random string $R = r_1 : r_2 : r_3 : \cdots : r_m$ of length $m \cdot L(n)$, for some specified $m \in \mathbb{N}$.

Algorithm 1 Algorithm $B_m(x, R)$

1: Let n denote the number of bits in x.
2: Break R into strings $r_1, r_2, \ldots, r_m$, each of length $L(n)$.
3: Let $a = \frac{1}{m} \sum_{i=1}^{m} A(x, r_i)$.
4: If $a \geq \frac{1}{2}$, output 1. Else, output 0.

Lemma 1.14. *If randomized algorithm $A(x, r)$ satisfies $\Pr(A(x, r) \neq \Pi(x)) \leq \frac{1}{3}$ for all x, then for any $\delta > 0$, randomized algorithm $B_m(x, R)$ with $m > 18 \ln(1/\delta)$ satisfies $\Pr(B_m(x, R) \neq \Pi(x)) \leq \delta$ for all x.*

Proof. Since algorithm A satisfies the BPP property (7), when $\Pi(x) = 1$ we have $\mathbb{E}[A(x, r_i)] \geq \frac{2}{3}$ and when $\Pi(x) = 0$ we have $\mathbb{E}[A(x, r_i)] \leq \frac{1}{3}$. If $B_m(x, R) \neq \Pi(x)$ then either $\Pi(x) = 0$ and $\sum_{i=1}^{m} A(x, r_i) \geq \frac{m}{2}$, or $\Pi(x) = 1$ and $\sum_{i=1}^{m} A(x, r_i) < \frac{m}{2}$. In the former case, $\sum_{i=1}^{m} A(x, r_i)$ exceeds its expected value by at least $\frac{m}{6}$, while in the latter case it falls short of its expected value by at least the same amount. In both cases, Hoeffding's Inequality ensures that the probability of this occurring is no greater than

$$e^{-2(m/6)^2/m} = e^{-m/18} < e^{\ln(\delta)} = \delta.$$

□

Lemma 1.14 has the following consequence for complexity theory. A decision problem Π is said to belong to the complexity class P/poly if there is a family of deterministic algorithms $\{B_n \mid n \in \mathbb{N}\}$ such that:

1. for every input x of size n, $B_n(x) = \Pi(x)$;
2. for some constant $c < \infty$ and every $n \in \mathbb{N}$, the worst-case running time of B_n on inputs of size n is bounded by $O(n^c)$.

This is summarized by saying that the decision problem Π has a *non-uniform family of polynomial-time algorithms*: it can be solved deterministically in polynomial time for all input sizes, but the choice of algorithm depends on the input size.

Theorem 1.15. *If Π is a decision problem in* BPP *then Π belongs to* P/poly.

Proof. Let A be a randomized polynomial-time algorithm for Π that satisfies property (7). For any $n \in \mathbb{N}$ let $m = \lceil 18 \ln(2) \cdot n \rceil = \lceil 18 \ln(2^n) \rceil$ and consider the randomized algorithm B_m. According to Lemma 1.14, for all $x \in \{0,1\}^n$, $\Pr(B_m(x, R) \neq \Pi(x)) < 2^{-n}$. By the union bound, $\Pr(\exists x \in \{0,1\}^n \; B_m(x, R) \neq \Pi(x)) < 1$. Hence, it is not the case that for all $R \in \{0,1\}^{m \cdot L(n)}$, there exists an $x \in \{0,1\}^n$ such that $B_m(x, R) \neq \Pi(x)$. In other words, there exists some $R_n \in \{0,1\}^{m \cdot L(n)}$ such that for all $x \in \{0,1\}^n$, $B_m(x, R_n) = \Pi(x)$. Let B_n be the algorithm that on input x, computes $B_m(x, R_n)$. Then the family $\{B_n \mid n \in \mathbb{N}\}$ constitutes a non-uniform family of polynomial-time algorithms for Π. □

A very natural and worthy goal is to eliminate the non-uniformity in Theorem 1.15 and prove that BPP = P. This would show that giving algorithms access to random bits does not affect the set of decision problems that can be solved in polynomial time, or equivalently, that every polynomial-time randomized algorithm for a decision problem can be efficiently "derandomized" to yield a deterministic polynomial-time algorithm for the same problem. Most complexity theorists believe such a derandomization of BPP is possible. The effort to derandomize BPP and other complexity classes is currently one of the most active research areas in complexity theory.

2 Hashing

In Section 1.3 we saw that throwing m balls at random into n bins is an excellent way to balance load. However, in CS the metaphorical "balls" — often pieces of data or tasks — may have *identifiers* or *keys*, and the event of throwing a ball with identifier x into bin b needs to be *reproducible*. In other words, the bin number b must be bound to the identifier x in such a way that the system can later fulfill a request to retrieve the ball with identifier x, or to remove it from its bin, or to throw another ball with the same identifier into the same bin.

The word for this type of "random but reproducible" mapping of balls to bins is *hashing*. In this section we will introduce an extremely important application of hashing in CS, namely dictionary data structures. In Section 2.1 we'll begin by defining the dictionary as an abstract data type. We'll describe a simple deterministic implementation of a dictionary and analyze its space and time efficiency. It turns out that all of the most efficient known implementations of dictionaries use randomization in the form of *hash functions*. In Section 2.2 we'll introduce the abstraction of hash functions. Then in Section 2.3 we'll introduce the hash table, a randomized data structure that uses hash functions to implement a dictionary. We'll first analyze the hash table's efficiency under an unrealistically strong assumption about the randomness of the hash function. Then in Section 2.4 we'll see how to substitute a weaker randomness assumption called *pairwise independence* without hurting the expected efficiency of the hash table. Furthermore, we'll present some efficient constructions of pairwise independent families of hash functions.

2.1 Introducing the dictionary data structure

The quintessential application of this capability is the *dictionary* abstract data type, also known as an associative array or key-value store. A data structure that implements a dictionary stores a set of key-value pairs, with at most one value per key. We will use $\mathcal{X}$ to denote the set of potential keys, and $x \in \mathcal{X}$ to denote one such key. Similarly we will use $\mathcal{V}$ to denote the set of potential values, and v to denote one such value. The dictionary supports (at least) the following three operations.

1. Lookup(x, v) returns the value v stored with key x, or it returns a special "not found" symbol, $\perp$, if x is not in the dictionary.
2. Insert(x, v) inserts the pair (x, v) into the dictionary. If another pair (x, v') was already stored in the dictionary, it is overwritten. (The value v' is replaced with v.)

3. DELETE(x) removes the (unique) pair (x, v) with key x from the dictionary, if any such pair exists.

Data structures implementing this functionality in common programming languages include the Python dictionary, the Java HashMap, and the C++ unordered_map. These implementations often provide other functionality (e.g., iterators) but in these notes we will focus on the three basic operations listed above, which are the three essential operations a dictionary must support.

A reasonably efficient deterministic implementation of the dictionary data type uses a balanced binary search tree, such as a red-black tree, to store the keys and pointers to the values. Then, all three of the dictionary operations can be implemented to run in $O(\log_2 m)$ time. As for the space efficiency of this dictionary implementation, if we assume each key and value occupies $O(1)$ space, then the storage required for this data structure is $O(m)$. More generally, if S is the total amount of storage occupied by all the keys and values (which might asymptotically exceed m, for example if the values stored in the dictionary are large objects that occupy a super-constant amount of space) then the balanced binary search tree, and the accompanying storage space for storing the values that the tree nodes point to, will occupy $O(S)$ space.

To sum up, the balanced binary search tree is already quite a space-efficient and time-efficient implementation of a dictionary: its space usage is within a constant factor of optimal, and the time complexity of each operation is $O(\log m)$. However, dictionaries are so widely used in computing, and their efficiency is so highly prized, that the running time of $O(\log m)$ per operation is considered slow. We will see that there is a randomized implementation with $O(1)$ running time per operation.

2.2 Hash functions

Hash tables are a randomized implementation of dictionaries in which the LOOKUP, INSERT, and DELETEoperations are faster than in the balanced binary search tree. They speed up the (expected) running time of these operations from $O(\log m)$ to $O(1)$ by "flattening" the data structure: instead of searching for keys stored in a hierarchical structure of depth $\log_2(m)$, the keys are stored in a one-dimensional array of $n = O(m)$ *hash buckets*, and a *hash function* applied to any key directs the user to the appropriate bucket.

A *hash function* needs to support two operations, generally with the aid of a source of random bits.

- INITHASH($\mathcal{X}, \mathcal{B}$) initializes a hash function with $\mathcal{X}$ as the set of potential keys and $\mathcal{B}$ as the set of bins. The operation is called only once, when the hash function is initialized.
- HASH(x) evaluates the hash function on key $x \in \mathcal{X}$, returning bin $b \in \mathcal{B}$.

Implementations of hash functions strive for an unattainable goal of achieving three properties simultaneously.

Uniform randomness If we call INITHASH($\mathcal{X}, \mathcal{B}$) followed by HASH(x_1), HASH(x_2), ..., HASH(x_m)

for any distinct keys $x_1, x_2, \ldots, x_m \in \mathcal{X}$, the values returned are independent, uniformly random elements of $\mathcal{B}$.

Reproducibility For any given $x \in \mathcal{X}$, all calls to $\textsc{Hash}(x)$ must return the same value.

Space and time efficiency The hash function should be stored using a small amount of space, and evaluating $\textsc{Hash}(x)$ should be fast. If $N = |\mathcal{X}|$, $n = |\mathcal{B}|$, and locations in memory can store $\log(n)$ bits, then the space required to store the representation of the hash function should ideally be $O(\log_2 N)$ and the time required to evaluate $\textsc{Hash}(x)$ should ideally be $O(\log_n N)$.

It's easy to achieve any two of these properties while sacrificing the third one. For example, the second and third properties are satisfied by any deterministic function that is easy to store and evaluate, for example a constant function that always outputs 1. The first and third properties are satisfied if we implement $\textsc{Hash}(x)$ by calling a random number generator to draw a uniformly random sample from $\mathcal{B}$ every time Hash is called, but of course this violates the second property (reproducibility). The first and second properties are achieved by implementing $\textsc{InitHash}(\mathcal{X}, \mathcal{B})$ to sample a uniformly random function $h : \mathcal{X} \to \mathcal{B}$ and store its values in a giant array of size N, but this is definitely not space-efficient, and the time-efficiency of evaluating $\textsc{Hash}(x)$ also suffers if N is large enough that an array of size N doesn't fit in RAM.

However, it is impossible to implement a hash function that achieves all three properties simultaneously. To see why, consider initializing a hash function with key set $\mathcal{X}$ and bucket set $\mathcal{B}$ and then calling

$$\textsc{Hash}(x_1), \textsc{Hash}(x_2), \ldots, \textsc{Hash}(x_m), \textsc{Hash}(x_1), \textsc{Hash}(x_2), \ldots, \textsc{Hash}(x_m)$$

where the keys $x_1, x_2, \ldots, x_m$ are any m distinct elements of $\mathcal{X}$. The implementation of the hash function has an internal state, and we will use s to represent the state at the end of the m^{th} call to Hash. For any $\mathbf{b} = (b_1, b_2, \ldots, b_m) \in [n]^m$ let $S(b)$ denote the set of internal states that may potentially be reached in an execution of the above sequence of Hash operations when the first m Hash operations have outputs $b_1, \ldots, b_m$ respectively. We now make the following observations.

1. By the uniform randomness property, every $\mathbf{b} \in [n]^m$ has a positive probability of being realized as the outputs of the first m Hash operations. Hence, $S(\mathbf{b})$ is non-empty for every $\mathbf{b} \in [n]^m$.

2. If $\mathbf{b} \neq \mathbf{b}'$ then $S(b)$ and $S(b')$ must be disjoint. To see why, for any internal state s and consider the outcome of calling $\textsc{Hash}(x_1), \textsc{Hash}(x_2), \ldots, \textsc{Hash}(x_m)$ starting from internal state s. By the reproducibility property, if $s \in S(\mathbf{b})$ then $\textsc{Hash}(x_1), \ldots, \textsc{Hash}(x_m)$ must return $\mathbf{b}$, whereas if $s \in S(\mathbf{b}')$ then $\textsc{Hash}(x_1), \ldots, \textsc{Hash}(x_m)$ must return $\mathbf{b}'$. Since $\mathbf{b} \neq \mathbf{b}'$, s cannot belong to both $S(\mathbf{b})$ and $S(\mathbf{b}')$, so the two sets are disjoint as claimed.

3. The set $\bigcup_{\mathbf{b} \in [n]^m} S(\mathbf{b})$ is a union of n^m disjoint non-empty sets, so its cardinality is at least n^m.

4. To encode each internal state using a distinct string of bits, we must use at least $\log_2(n^m) = m \log_2(n)$ bits to encode the internal state.

5. A single memory location holds at most $\log_2(n)$ bits. Hence, storing the internal state requires at least m memory locations. When $m \gg \log(N)$ this violates space-efficiency.

2.3 Hash tables

In applications of hashing, reproducibility is usually treated as a hard constraint, and space and time efficiency are strongly desired. (The more efficient, the better.) On the other hand, the property of uniform randomness is often stronger than what's really needed. A useful paradigm for designing algorithms that make use of hashing is to start by fantasizing that all three of the properties we desire in a hash function can be fulfilled. Under this unrealistic assumption, we design and analyze an algorithm. Then, we scrutinize the analysis of the algorithm to identify a weakening of the uniform randomness property that suffices for the analysis. Lastly, we try to design a hash function implementation that satisfies the weaker property while maintaining space and time efficiency.

An important and illustrative case study is the *hash table with chain hashing*, which implements a dictionary using a randomized hash function to map keys to buckets. Suppose that, when initializing the dictionary, we are given the space of potential keys, $\mathcal{X}$, the space of potential values, $\mathcal{V}$, and an upper bound on the maximum number of key-value pairs[2] that will ever be stored in the dictionary at once, m. Given these parameters, the hash table uses a set $\mathcal{B} = [n]$ of $n = O(m)$ hash buckets, each with an associated linked list where key-value pairs are stored. A hash function is used to map keys to bins. The hash table operations are implemented as follows.

1. To initialize the hash table, one calls InitHash($\mathcal{X}, \mathcal{B}$), allocates an array of size n for the bins, and populates each cell of the array with a pointers to an empty linked list.

2. Lookup(x) calls Hash(x) to obtain a bucket b, then searches the linked list stored in bucket b to see if a pair (x, v) is found.

3. Insert(x, v) first calls Lookup(x). If a pair (x, v') is found, the value stored is modified from v' to v. If x is not found in the hash table, the pair (x, v) is appended to the linked list stored in bucket Hash(x).

4. Delete(x) scans the linked list stored in bucket Hash(x) and, if a pair with key x is found, deletes that pair from the linked list.

On an architecture where keys and values can be stored in constant space, the space required for the hash table is $O(m+n)$. The time required for Lookup, Insert, and Deleteoperations is linear in the length of the linked list stored in the bucket Hash(x), where x is the key associated with the operation.

[2]For the sake of convenience, we are assuming that an upper bound on m is known at initialization time. If this assumption is not satisfied, there are "doubling tricks" that involve making an initial guess that $m = O(1)$, and then every time the number of key-value pairs stored in the hash table exceeds the current guess, the guess is doubled and the hash table is resized accordingly. One then needs to analyze the time complexity of these resizing operations. It is not hard to show that the amortized cost of resizing the hash table is bounded by the total cost of the insertion operations. We omit the amortized analysis from these notes, but it can found in textbooks on data structures.

Suppose for a moment that the hash function is a uniformly random function $h : \mathcal{X} \to \mathcal{B}$. Then, we could bound the expected length of the linked list stored in the bucket $b = \textsc{Hash}(x)$ as follows. If x itself is already stored in the hash table, then b definitely contains one element. There are at most $m - 1$ other elements $y \neq x$ stored in the hash table, and each of them has probability $1/n$ of being stored in bucket b, so the expected length of the linked list in that bucket is $1 + (m-1)/n$. If x is not stored in the hash table, then there are at most m elements stored in it, each of them has $1/n$ probability of belonging to bucket b, and hence the expected length of the linked list is m/n. The ratio m/n is called the *load factor* of the hash table. Our analysis has shown that *if the hash function is uniformly random*, the expected running time of hash table operations is $O(1 + m/n)$. Typically one chooses the parameters of the hash table to make the load factor a constant less than 1, resulting in $O(1)$ running time for lookup, insertion, and deletion.

It would appear that this entire analysis rests on the assumption that the hash function is uniformly random, an assumption which unfortunately is incompatible with space and time efficiency of the hash function operations. Fortunately, upon closer examination, our analysis only made use of the randomness of h in one step, when bounding the expected number of elements other than x that occupy bucket $b = \textsc{Hash}(x)$. By linearity of expectation, this quantity can be calculated as

$$\sum_{b \in \mathcal{B}} \sum_{y \neq x} \Pr(\textsc{Hash}(x) = \textsc{Hash}(y) = b).$$

The event $\textsc{Hash}(x) = \textsc{Hash}(y) = b$ is called a *hash collision* of keys x and y at bucket b. For any specific x, y, and b, the probability of the event $\textsc{Hash}(x) = \textsc{Hash}(y) = b$ is $1/n^2$. The double-sum above has at most mn terms, each equal to $1/n^2$, so the expected number of $y \neq x$ occupying bucket $b = \textsc{Hash}(x)$ is bounded above by $(mn)/n^2 = m/n$, the load factor of the hash table.

2.4 Pairwise independence

Reviewing the analysis of the hash table with chain hashing, we see that the hash function need not be uniformly random, it only needs to satsfy the much weaker property that for all $b \in \mathcal{B}$ and all $x, y \in \mathcal{X}$ such that $x \neq y$, we have $\Pr(\textsc{Hash}(x) = \textsc{Hash}(y) = b) = 1/n^2$.

Definition 2.1. A *2-universal hash family* is a set of functions $\mathcal{H}$, with each $h \in \mathcal{H}$ being a function from $\mathcal{X}$ to $\mathcal{B}$, such that when h is sampled uniformly at random from $\mathcal{H}$, for every distinct $x, y \in \mathcal{X}$ the ordered pair of values $(h(x), h(y))$ is uniformly distributed over $\mathcal{B}^2$.

A 2-universal hash family is sometimes also called a *pairwise independent hash family*, though that term is a bit of a misnomer because the definition requires not only that pair of the hash values $h(x)$ and $h(y)$ are independent, but also that each of them is uniformly distributed.

Since the analysis of chain hashing in Section 2.3 relied only on the 2-universality of the hash family, we have established the following.

Proposition 2.1. *Consider a hash table with n buckets that uses chain hashing with a hash function drawn uniformly from a 2-universal hash family. Let $s_{\text{hash}}(n)$ denote the space complexity of storing a representation of the hash function, and let $t_{\text{hash}}(n)$ denote the time complexity of evaluating* $\textsc{Hash}(x)$ *on any given key x. For any sequence of hash table operations with at most $m \leq cn$ key-value pairs*

stored in the table at any time, the expected time per operation is $O(c+1+t_{\text{hash}})$. *The space complexity of the hash table is* $O(m+n+s_{\text{hash}})$.

Of course, in order for a 2-universal hash family to be computationally useful, it is necessary to be able to efficiently store and evaluate the hash functions in the family. The remainder of this section provides some useful constructions of 2-universal hash families.

2.4.1 Linear congruential hashing

The simplest construction of a 2-universal hash function works when the number of hash buckets is a prime number, p, and the space of potential keys, $\mathcal{X}$, has p or fewer[3] elements.

Let $\mathbb{F}_p$ denote the set $\{0, 1, \ldots, p-1\}$ under the operations of addition and multiplication modulo p. In other words, to add or multiply two elements of $\mathbb{F}_p$, one adds them or multiplies them as ordinary integers, and then if the result is greater than or equal to p, one divides it by p and outputs the remainder. This structure is called the *prime field of order p*, but for present purposes it is not necessary to know what a field is in order to analyze the hash function family we now present.

The *linear congruential hash function family modulo p* is the family of functions $h : \mathbb{F}_p \to \mathbb{F}_p$ defined by

$$h(x) = ax + b$$

where the coefficients a and b are allowed to be any elements of $\mathbb{F}_p$. Denote this family of functions by $\mathcal{H}_p$.

Lemma 2.2. *For any prime number p, the linear congruential hash function family* $\mathcal{H}_p$ *is a 2-universal hash family.*

Proof. The hash function family $\mathcal{H}_p$ has p^2 elements, one for each pair of coefficients $a, b \in \mathbb{F}_p$. We intend to argue that for any $x \neq y$, the function $\phi_{x,y} : \mathcal{H}_p \to \mathbb{F}_p^2$ defined by

$$\phi_{x,y}(h) = (h(x), h(y))$$

is a bijection. If so, then it follows that for h drawn uniformly at random from $\mathcal{H}_p$, the pair $(h(x), h(y)) = \phi_{x,y}(h)$ will be drawn uniformly at random from $\mathbb{F}_p^2$ as required by the definition of a 2-universal hash family.

Since $\mathcal{H}_p$ and $\mathbb{F}_p^2$ both have exactly p^2 elements, the assertion that $\phi_{x,y}$ is a bijection is equivalent to the assertion it is one-to-one. In other words, if h, h' are two (possibly identical) elements of $\mathcal{H}_p$ such that $\phi_{x,y}(h) = \phi_{x,y}(h')$, we are accountable for proving that $h = h'$. Let a, b and a', b' be the coefficient pairs for h and h', respectively. In other words, for all z, $h(z) = az + b$ and

[3] In typical hash tables, the number of potential keys is *much* greater than the number of buckets, so the assumption that $|\mathcal{X}| \leq p$ is quite limiting. The hash function family that we construct and analyze in this section nevertheless has other very useful applications. One such application will be presented later in Section 3.2.

$h'(z) = a'z + b'$. Then, rewriting the equation $\phi_{x,y}(h) = \phi_{x,y}(h')$ as $h(x) = h'(x)$ and $h(y) = h'(y)$ we find that

$$\begin{aligned}(a - a')x + (b - b') &= h(x) - h'(x) = 0\\ (a - a')y + (b - b') &= h(y) - h'(y) = 0\\ (a - a')(x - y) &= 0\end{aligned}$$

where all addition and multiplication operations are interpreted modulo p, and the equation on the last line is obtained by subtracting both sides of the equations on the two lines above. The equation $(a-a')(x-y) = 0$ (modulo p) means that p is a divisor of the product $(a-a')(x-y)$. Since p is prime, it must divide at least one of the factors, $a-a'$ or $x-y$. However, since a, a', x, y all belong to the set $\{0, 1, \ldots, p-1\}$, the differences $a - a'$ and $x - y$ belong to the interval $[-(p-1),\ p-1]$, and 0 is the only multiple of p in that interval. Hence, either $a = a'$ or $x = y$. By assumption $x \neq y$, so we have shown $a = a'$. Now, rewriting the equation $h(x) = h'(x)$ as $ax + b = a'x + b'$ and using the equation $a = a'$, we find that $b = b'$ as well. Thus, $h = h'$ as desired. □

To evaluate the space and time efficiency of linear congruential hashing, recall our standing assumption that one memory location can store $O(\log n) = O(\log p)$ bits, which is sufficient to store the value of one element of $\mathbb{F}_p$. To store a representation of a hash function $h \in \mathcal{H}_p$ one only needs to store the coefficients a and b, hence the representation of the function h requires only $O(1)$ space. Evaluating h requires only 2 arithmetic operations (mod p), so it takes $O(1)$ time.

2.4.2 Inner product hashing

Linear congruential hashing can be generalized to allow for applications in which the number of buckets is still a prime number, p, but the number of potential keys, N, may be much greater than p. In that case, we will let $d = \lceil \log_p(N) \rceil$ and we will identify the space of N keys, $\mathcal{X}$, with a subset of the set $\mathbb{F}_p^d$ of d-tuples of elements of $\mathbb{F}_p$. The set $\mathbb{F}_p^d$ constitutes a *d-dimensional vector space over the prime field of order p*. Analogous to the dot-product operation on vectors in $\mathbb{R}^d$ we have the following *inner product* operation which is defined for any two elements $\mathbf{w} = (w_1, \ldots, w_d)$ and $\mathbf{x} = (x_1, \ldots, x_d)$ in $\mathbb{F}_p^d$.

$$\langle \mathbf{w}, \mathbf{x} \rangle = \sum_{i=1}^{d} w_i x_i.$$

As usual, the addition and multiplication operations on the right side are interpreted modulo p. Now, let $\mathcal{H}_p^d$ denote the set of hash functions $h : \mathbb{F}_p^d \to \mathbb{F}_p$ of the form

$$h(\mathbf{x}) = \langle \mathbf{a}, \mathbf{x} \rangle + b$$

where $\mathbf{a} \in \mathbb{F}_p^d$ and $b \in \mathbb{F}_p$. Storing the representation of a function $h \in \mathcal{H}_p^d$ requires storing $d + 1$ elements of $\mathbb{F}_p$, which takes $O(d)$ space. Evaluating $h(\mathbf{x})$ takes $O(d)$ time, since it involves performing d multiplication and d addition operations in $\mathbb{F}_p$.

Lemma 2.3. *For any prime number p, the linear congruential hash function family $\mathcal{H}_p$ is a 2-universal hash family.*

Proof. We can prove that the hash family $\mathcal{H}_p^d$ is 2-universal by mimicking the proof in the $d = 1$ case, Lemma 2.3. Consider any $\mathbf{x}, \mathbf{y} \in \mathbb{F}_p^d$. If $\mathbf{x} \neq \mathbf{y}$ it means there is a coordinate j such that $x_j \neq y_j$. Without loss of generality assume $j = 1$. Then, we can break down the process of sampling $h \in \mathcal{H}_p^d$ into two steps:

1. sample $a_2, a_3, \ldots, a_d \in \mathbb{F}_p$ independently and uniformly at random;
2. sample $a_1,\ b \in \mathbb{F}_p$ independently and uniformly at random.

For any fixed choice of $a_2, \ldots, a_d$ in the first sampling step, there are p^2 choices of a_1 and b in the second step and corresponding to each of them there is an ordered pair of hash values $(h(\mathbf{x}), h(\mathbf{y})) \in \mathbb{F}_p^2$. If we can prove that the mapping $(a_1, b) \mapsto (h(\mathbf{x}), h(\mathbf{y}))$ is one-to-one, then it must be bijective, from which it follows that $(h(\mathbf{x}), h(\mathbf{y}))$ is uniformly distributed over $\mathbb{F}_p^2$, conditional on our fixed choice of $a_2, \ldots, a_d$. Since this holds for any fixed choice of $a_2, \ldots, a_d$, it then follows (by averaging over $a_2, \ldots, a_d$) that the unconditional distribution of $(h(\mathbf{x}), h(\mathbf{y}))$ is uniform over $\mathbb{F}_p^2$, as desired.

To prove that the function $(a_1, b) \mapsto (h(\mathbf{x}), h(\mathbf{y}))$ is one-to-one, consider any coefficient pairs (a_1, b) and (a_1', b') with associated hash functions h and h', respectively, and assume $h(\mathbf{x}) = h'(\mathbf{x})$ and $h(\mathbf{y}) = h'(\mathbf{y})$. Define $\mathbf{a} = (a_1, a_2, \ldots, a_d)$ and $\mathbf{a}' = (a_1', a_2, \ldots, a_d)$; note that these two vectors differ only in their first coordinate. We have

$$\begin{aligned}
0 &= h(\mathbf{x}) - h'(\mathbf{x}) = \langle \mathbf{a}, \mathbf{x} \rangle + b - \langle \mathbf{a}', \mathbf{x} \rangle - b' = \langle \mathbf{a} - \mathbf{a}', \mathbf{x} \rangle + (b - b') = (a_1 - a_1')x_1 + (b - b') \\
0 &= h(\mathbf{y}) - h'(\mathbf{y}) = \langle \mathbf{a}, \mathbf{y} \rangle + b - \langle \mathbf{a}', \mathbf{y} \rangle - b' = \langle \mathbf{a} - \mathbf{a}', \mathbf{y} \rangle + (b - b') = (a_1 - a_1')y_1 + (b - b') \\
0 &= (a_1 - a_1')(x_1 - y_1).
\end{aligned}$$

By assumption, $x_1 \neq y_1$ so, as in the proof of Lemma 2.3, it follows that $a_1 = a_1'$. In that case $\mathbf{a} = \mathbf{a}'$, so rewriting the equation $h(\mathbf{x}) = h'(\mathbf{x})$ as $\langle \mathbf{a}, \mathbf{x} \rangle + b = \langle \mathbf{a}, \mathbf{x} \rangle + b'$, we see that it implies $b = b'$ and hence $h = h'$, as desired. □

3 Data Streaming and Sketching

In this section we survey some of the applications of hashing to the analysis of datasets that are too large to fit in the computer's memory all at once.

In the *streaming* model of computation, an algorithm observes a sequence $a_1, a_2, \ldots, a_n$ of data items, each represented by at most b bits. Thus, the set of potential data items (called "tokens" henceforth) has size $m = 2^b$. The algorithm has a working memory of size s, where each memory location is assumed to be capable of storing $O(\log n)$ bits. Typically we require s to have sublinear dependence on n (the length of the stream) and at most linear dependence on b (the number of bits representing each element). Hence it is infeasible to store each data item, which would require space $s \geq n$ even if b were $O(\log n)$, and it's also infeasible to store a count of how many times each token was seen in the data stream, which could require space $s \geq 2^b$ if every element of $\{0, 1\}^b$ were observed at least once in the stream.

Some of the typical objectives of streaming algorithms are to find the most frequently occurring element (or elements) in the data stream, approximate the number of distinct elements, or approximate the p^{th} frequency moment, $\sum_j f_j^p$, where f_j denotes the number of occurrences of the token j in the stream.

3.1 Finding frequent elements

To illustrate the model, we begin by presenting an example of a non-trivial streaming algorithm that makes no use of hashing and is, in fact, completely deterministic. This is an algorithm of Misra and Gries that uses space $s = O(k(b + \log n))$ to find every token that occurs more than $n/(k+1)$ times in the stream. The algorithm allocates its storage space for a k-tuple of tokens $b_1, \ldots, b_k$, and a k-tuple of counters, $c_1, \ldots, c_k$. Initially each pair (b_j, c_j) is initialized to $(\perp, 0)$, where $\perp$ denotes a null symbol that doesn't belong to the set of tokens. While the algorithm is processing the stream, if it sees one of the tokens $b_1, \ldots, b_k$ then it increments the corresponding counter. Otherwise, if one of the counters c_j is equal to zero, it stores the new element as b_j and sets c_j to 1. Otherwise, if all of the counters are strictly positive, it decrements each of them. When the algorithm finishes processing the stream, it outputs the set of all tokens that have positive counters.

Initialize $(b_j, c_j) = (\perp, 0)$ for $j = 1, 2, \ldots, k$.
for $i = 1, 2, \ldots, n$ **do**
    if $a_i = b_j$ for some $j \in [k]$ **then**
        $c_j \leftarrow c_j + 1$
    else if $c_j = 0$ for some $j \in [k]$ **then**
        $b_j \leftarrow a_i$
        $c_j \leftarrow 1$
    else
        Decrement c_j to $c_j - 1$ for each $j \in [k]$.
    end if
end for
Output $\{b_j \mid c_j > 0\}$.

Proposition 3.1. *The output of the Misra-Gries algorithm contains every token that occurs more than $n/(k+1)$ times in the data stream (and potentially some tokens that occur fewer than $n/(k+1)$ times).*

Proof. Picture marking elements of the sequence $a_1, a_2, \ldots, a_n$ as follows. Initially all elements are unmarked. At the start of the loop iteration that processes element a_i, it becomes marked. There are three cases for what could happen during the loop iteration. In the first two cases, if $a_i \in \{b_1, \ldots, b_k\}$ or if $a_i \notin \{b_1, \ldots, b_k\}$ but $c_j = 0$ for some j, then a_i remains marked. In the third case, if $a_i \notin \{b_1, \ldots, b_k\}$ and $c_j > 0$ for all j, then we remove the mark from a_i, and we also remove red marks from the earliest marked copy of each of the tokens $b_1, \ldots, b_k$.

We claim that at all times, there are c_j marked copies of b_j for each $j \in [k]$, and no token other than $b_1, \ldots, b_k$ is marked. The proof is by induction on i. In the base case $i = 0$, no tokens are marked and $c_j = 0$ for all j. For the induction step, if a_i belongs to the set $\{b_1, \ldots, b_k\}$ or is inserted into that set, then it remains marked at the end of the loop iteration and the corresponding counter c_j is incremented. If a_i doesn't belong to the set $\{b_1, \ldots, b_k\}$ and $c_j > 0$ for all j, then the mark is removed from a_i and (by the induction hypothesis) there is at least one marked copy of b_j for every $j \in [k]$, so a mark is removed from one copy of each b_j as c_j is decremented.

Each time a loop iteration removes any marks, it removes $k + 1$ of them. Since an element of the sequence is only marked once and its mark is removed at most once, there are at most $n/(k + 1)$ loop iterations in which marks are removed. If a token appears strictly more than $n/(k + 1)$ times in the sequence, then some copies of that token are marked at the end of the final loop iteration, so that token must be one of $b_1, \ldots, b_k$. □

3.2 Estimating the number of distinct elements

The Misra-Gries algorithm is atypical of streaming algorithms because it's deterministic. Generally a streaming algorithm's objective can't be achieved deterministically within the given space bound, so these algorithms use randomness and are usually evaluated according to the PAC (probably approximately correct) objective: one wants to show that with probability at least $1 - \delta$, the algorithm's output approximates the target quantity with relative error ε or less.

Here's a famous example due to Flajolet and Martin. The algorithm estimates the number of distinct tokens in the data stream. Note that this number might be as large as $m = 2^b$, but we aim to estimate the number of distinct token in space $s = \text{poly}(b, \log n)$, so keeping a list of every distinct token encountered in the stream is not an option. Instead, we will use a hash function $h : [m] \to [M]$, for some large integer M. The function h will be drawn at random from a 2-universal hash family.

The key observation is that if there are d distinct tokens in the stream, then the random variable $Z = \min\{h(a_i) \mid 1 \le i \le n\}$ is on the order of $\frac{M}{d}$. In fact, if we assume without loss of generality that the d distinct tokens belonging to the stream are $a_1, \ldots, a_d$, then for any $k \in [M]$ we can define the random variables

$$X_{ik} = \begin{cases} 1 & \text{if } h(a_i) \le k \\ 0 & \text{otherwise} \end{cases}$$

$$Y_k = \sum_{i=1}^{d} X_{ik} = \text{number of distinct tokens whose hash value is } \le k.$$

Then, we make the following observations.

1. $\mathbb{E}[X_{ik}] = \frac{k}{M}$.
2. $\mathbb{E}[Y_k] = \frac{dk}{M}$.

3. $\mathrm{Var}[Y_k] = \frac{dk(M-k)}{M^2} < \frac{dk}{M}$. This is because

$$\begin{aligned}
\mathrm{Var}[Y_k] = \mathbb{E}[Y_k^2] - \mathbb{E}[Y_k]^2 &= \sum_{i=1}^{d}\sum_{j=1}^{d} \mathbb{E}[X_{ik}X_{jk}] - \frac{d^2k^2}{M^2} \\
&= \sum_{i=1}^{d} \mathbb{E}[X_{ik}] + \sum_{i=1}^{d}\sum_{j\neq i} \mathbb{E}[X_{ik}X_{jk}] - \frac{d^2k^2}{M^2} \\
&= d\cdot\frac{k}{M} + d(d-1)\cdot\frac{k^2}{M^2} - -\frac{d^2k^2}{M^2} \\
&= d\cdot\frac{k}{M} - d\cdot\frac{k^2}{M^2} = \frac{dk(M-k)}{M^2}.
\end{aligned}$$

In these observations, we made use of the assumption that the hash function h is sampled from a 2-universal hash family. The first such usage was when we asserted $\mathbb{E}[X_{ik}] = \frac{k}{M}$, which requires $h(a_i)$ to be uniformly distributed. The second such usage was in the calculation of $\mathrm{Var}[Y_k]$: the derivation of the third line uses the fact that $\mathbb{E}[X_{ik}X_{jk}] = \frac{k^2}{M^2}$, which follows from the pair $(h(a_i), h(a_j))$ being uniformly distributed in $[M]^2$.

Recall that $Z = \min\{h(a_i) \mid i \in [n]\}$. As a first attempt at estimating d, we can approximate it with the quantity M/Z. By Markov's Inequality, if $k = \lfloor M/6d \rfloor$, then

$$\Pr\left(\tfrac{M}{Z} > 6d\right) = \Pr\left(\tfrac{M}{6d} > Z\right) = \Pr\left(Y_k \geq 1\right) \leq \mathbb{E}[Y_k] = \tfrac{dk}{M} \leq \tfrac{1}{6}. \tag{8}$$

On the other hand, by Chebyshev's Inequality, if $\ell = \lfloor 6M/d \rfloor$,

$$\begin{aligned}
\Pr\left(\tfrac{M}{Z} < \tfrac{d}{6}\right) = \Pr\left(\tfrac{6M}{d} \leq Z\right) &= \Pr(Y_\ell = 0) \leq \Pr(|Y_\ell - \mathbb{E}Y_\ell| \geq \mathbb{E}Y_\ell) \\
&\leq \frac{\mathrm{Var}(Y_\ell)}{(\mathbb{E}Y_\ell)^2} \\
&< \frac{\mathbb{E}Y_\ell}{(\mathbb{E}Y_\ell)^2} = \frac{1}{\mathbb{E}Y_\ell} = \frac{M}{d\ell} \leq \frac{1}{6} + \frac{1}{6M-5}.
\end{aligned} \tag{9}$$

Hence, the probability that the estimate M/Z lies outside the interval $[d/6,\ 6d]$ is at most $\frac{1}{3} + \frac{1}{6M-5}$.

We can obtain a better estimate of d using Z_t, the t^{th} smallest of the values $\{h(a_i)\}_{i=1}^n$, for a suitable choice of the parameter $t > 1$. Intuitively, the reason is that Z_t "aggregates a greater amount of randomness", namely the randomness in the positions of the t smallest elements rather than just the smallest one. To make this intuition a bit more precise, if we set $k = \lfloor tM/d \rfloor$ such that the expected number of elements that hash into the set $[k]$ is $\mathbb{E}[Y_k] = dk/M \approx t$, then the variance $\mathrm{Var}[Y_k]$ is less than t, so the probability that Y_k differs from its expected value by more than εt is at most $\frac{1}{\varepsilon^2 t}$ by Chebyshev's Inequality. For $t > \frac{1}{\varepsilon^2\delta}$, this probability will be less than δ. This argument doesn't directly lead to the conclusion that tM/Z_t approximates d within ε, but a variation on the argument — using random variables Y_q and Y_r for values q and r differing from k by $1 \pm \varepsilon$ factors, and accounting more carefully for hash collisions — does the trick.

Algorithm 2 Algorithm for estimating distinct elements

Set $t = \lceil \frac{12}{\varepsilon^2 \delta} \rceil$.
Choose $M \geq 6m/(\varepsilon\delta)$ and randomly sample $h : [m] \to [M]$ from a 2-universal hash family.
Initialize $(Z_1, Z_2, \ldots, Z_t) = \perp^t$.
for $i = 1, \ldots, n$ **do**
    Observe a_i and calculate $z = h(a_i)$.
    if $z < Z_t$ **then**
        Update $Z_1, \ldots, Z_t$ to be the t smallest hash values yet seen, in increasing order.
    end if
end for
Output tM/Z_t.

Note that the space required by the algorithm is equal to $O(t \log M)$ plus the amount of space required to store h. In Section 2.4 we saw that when M is a prime number, we can sample h from the linear congruential hash family and the space required for storing h will be $O(\log M)$ bits. Since there is always a prime number between $6m/(\varepsilon\delta)$ and $12m/(\varepsilon\delta)$, we can ensure $\log(M) \leq \log(m) - \log(\varepsilon\delta) + O(1)$. Also $t \leq \frac{12}{\varepsilon^2\delta} + 1$, so the space required by the algorithm is $s = O\left(\frac{\log m - \log(\varepsilon\delta)}{\varepsilon^2 \delta}\right)$ bits.

Proposition 3.2. *When Algorithm 2 is run on a stream with d distinct elements, the probability that it outputs an answer in the range $[(1-\varepsilon)d, (1+\varepsilon)d]$ is at least $1 - \delta$.*

Proof. If the output, tM/Z_t, lies outside the range $[(1-\varepsilon)d, (1+\varepsilon)d]$, it means that Z_t lies outside the range $\left[\frac{tM}{(1+\varepsilon)d}, \frac{tM}{(1-\varepsilon)d}\right]$. To reason about the circumstances under which Z_t could lie outside the range $\left[\frac{tM}{(1+\varepsilon)d}, \frac{tM}{(1-\varepsilon)d}\right]$, we let

$$q = \left\lceil \frac{tM}{(1+\varepsilon)d} \right\rceil - 1, \qquad r = \left\lfloor \frac{tM}{(1-\varepsilon)d} \right\rfloor$$

and observe that there are three cases in which Z_t lies outside $\left[\frac{tM}{(1+\varepsilon)d}, \frac{tM}{(1-\varepsilon)d}\right]$.

1. If $Z_t < \frac{tM}{(1+\varepsilon)d}$ it means that at least t distinct tokens in the stream are mapped to hash buckets in the range $[q] = \{1, 2, \ldots, q\}$. Denote this event by $\mathcal{E}_1$.
2. If $Z_t > \frac{tM}{(1-\varepsilon)d}$ it means that one of the following two cases must occur.
    (a) Fewer than $\left(1 + \frac{\varepsilon}{2-2\varepsilon}\right)t$ distinct tokens in the stream are mapped to hash buckets in the range $[r]$. Denote this event by $\mathcal{E}_{2a}$.
    (b) At least $\left(1 + \frac{\varepsilon}{2-2\varepsilon}\right)t$ distinct tokens in the stream are mapped to hash buckets in $[r]$, but these tokens occupy fewer than t distinct buckets because there are more than $\frac{\varepsilon t}{2-2\varepsilon}$ hash collisions in buckets whose number belongs to $[r]$. Denote this event by $\mathcal{E}_{2b}$.

By the union bound, to prove $\Pr\left(Z_t \in \left[\frac{tM}{(1+\varepsilon)d}, \frac{tM}{(1-\varepsilon)d}\right]\right) \geq 1 - \delta$, it suffices to prove $\Pr(\mathcal{E}_1) + \Pr(\mathcal{E}_{2a}) + \Pr(\mathcal{E}_{2b}) \leq \delta$. We will do so by defining some random variables, reasoning about their expectations

and variances using the 2-universality of the random hash function h, and applying Markov's and Chebyshev's Inequalities.

Assume without loss of generality that $a_1, a_2, \ldots, a_d$ are the distinct tokens in the stream and for $i \in [d]$ and $\ell \in [M]$ let

$$X_{i\ell} = \begin{cases} 1 & \text{if } h(a_i) = \ell \\ 0 & \text{otherwise.} \end{cases}$$

The random variable $X_{i\ell}$ indicates whether a_i hashes into bucket ℓ, and for $i \neq j$ the random variable $X_{i\ell}X_{j\ell}$ indicates whether a_i and a_j collide in bucket ℓ. By 2-universality,

$$\mathbb{E}[X_{i\ell}] = \tfrac{1}{M} \tag{10}$$

$$\mathbb{E}[X_{i\ell}X_{j\ell}] = \tfrac{1}{M^2} \qquad \text{if } i \neq j. \tag{11}$$

For any $k \in [M]$, the random variable

$$Y_k = \sum_{i=1}^{d} \sum_{\ell=1}^{k} X_{i\ell}$$

counts the number of distinct tokens that hash to buckets numbered in the range $[k]$. Its expectation and variance can be calculated using equations (10) and (11).

$$\mathbb{E}[Y_k] = \frac{dk}{M} \tag{12}$$

$$\mathrm{Var}[Y_k] = dk\left(\frac{1}{M} - \frac{1}{M^2}\right) < \frac{dk}{M}. \tag{13}$$

We can use these calculations to bound the probabilities of $\mathcal{E}_1$ and $\mathcal{E}_{2a}$.

$$\Pr(\mathcal{E}_1) = \Pr(Y_q \geq t) \leq \Pr\left(|Y_q - \mathbb{E}[Y_q]| \geq t - \mathbb{E}[Y_q]\right) \leq \frac{\mathrm{Var}[Y_q]}{(t - \mathbb{E}[Y_q])^2} < \frac{dq/M}{(t - dq/M)^2}$$

$$\begin{aligned} \Pr(\mathcal{E}_2) &= \Pr\left(Y_r < \left(1 + \tfrac{\varepsilon}{2-2\varepsilon}\right)t\right) \leq \Pr\left(|Y_r - \mathbb{E}[Y_r]| > \mathbb{E}[Y_r] - \left(1 + \tfrac{\varepsilon}{2-2\varepsilon}\right)t\right) \\ &\leq \frac{\mathrm{Var}[Y_r]}{\left(\mathbb{E}[Y_r] - \left(1 + \frac{\varepsilon}{2-2\varepsilon}\right)t\right)^2} < \frac{dr/M}{\left(dr/M - \left(1 + \frac{\varepsilon}{2-2\varepsilon}\right)t\right)^2}. \end{aligned}$$

By our choice of q and t we have

$$\begin{aligned} \tfrac{dq}{M} &< \tfrac{t}{1+\varepsilon} \\ t - \tfrac{dq}{M} &> \tfrac{\varepsilon t}{1+\varepsilon} \\ \frac{dq/M}{(t - dq/M)^2} &< \frac{t/(1+\varepsilon)}{\varepsilon^2 t^2/(1+\varepsilon)^2} = \frac{1+\varepsilon}{\varepsilon^2 t} < \frac{\delta}{3}. \end{aligned}$$

By our choice of r and t we have

$$\begin{aligned} \tfrac{t}{1-\varepsilon} \leq \tfrac{dr}{M} &< \tfrac{t}{1-\varepsilon} + 1 \\ \tfrac{dr}{M} - \left(1 + \tfrac{\varepsilon}{2-2\varepsilon}\right)t &\geq \tfrac{t}{1-\varepsilon} - t - \tfrac{\varepsilon t}{2-2\varepsilon} = \tfrac{\varepsilon t}{2-2\varepsilon} \\ \frac{dr/M}{\left(dr/M - \left(1 + \frac{\varepsilon}{2}\right)t\right)^2} &< \frac{t/(1-\varepsilon) + 1}{\varepsilon^2 t^2/4(1-\varepsilon)^2} = \frac{4(1-\varepsilon) + 4(1-\varepsilon)^2/t}{\varepsilon^2 t} < \frac{\delta}{3} \end{aligned}$$

Hence, $\Pr(\mathcal{E}_1)$ and $\Pr(\mathcal{E}_{2a})$ are both less than $\delta/3$. To deal with $\Pr(\mathcal{E}_{2b})$ we observe that the number of pairs of distinct stream tokens whose hash values collide in bucket set $[r]$ is equal to the random variable $W = \sum_{\ell=1}^{r} \sum_{1 \le i < j \le d} X_{i\ell} X_{j\ell}$. By equation (11) and our choice of $M \ge \frac{6m}{\varepsilon\delta}$ we have

$$\mathbb{E}[W] = \frac{r \cdot \sharp\{(i,j) \mid 1 \le i < j \le d\}}{M^2} = \frac{rd(d-1)}{2M^2} < \left(\frac{dr}{M}\right)\left(\frac{d}{2M}\right) < \left(\frac{t+1}{1-\varepsilon}\right)\left(\frac{m}{2M}\right)$$

$$\Pr(\mathcal{E}_3) = \Pr\left(W > \frac{\varepsilon t}{2(1-\varepsilon)}\right) < \frac{\frac{t+1}{1-\varepsilon} \cdot \frac{m}{2M}}{\varepsilon t/2(1-\varepsilon)} = \left(1 + \frac{1}{t}\right)\frac{m}{\varepsilon M} < \frac{\delta}{3}.$$

We have shown that each of the events $\mathcal{E}_1$, $\mathcal{E}_{2a}$, $\mathcal{E}_{2b}$ has probability less than $\delta/3$, and that the union of these three events covers all cases in which tM/Z_t lies outside the interval $[(1-\varepsilon)d, (1+\varepsilon)d]$. Hence, with probability at least $1-\delta$, the algorithm's estimate tM/Z_t belongs to the interval $[(1-\varepsilon)d, (1+\varepsilon)d]$ as claimed. □

3.3 Sketching token frequencies

Data sketching is an algorithmic paradigm that combines streaming with data structures. As before, an algorithm processes a stream of tokens, $a_1, \ldots, a_n$, taking values in $[m]$, and it is allowed to store $s = O(\text{poly}(\log n, \log m))$ bits of information about the stream. However, rather than wanting to estimate a single attribute of the stream, such as the number of distinct elements, the algorithm designer's objective is to be able to answer queries about the stream afterward. In this setting, the s-bit internal representation of the stream is called a *sketch* of the data.

Consider the task of sketching the frequency of each token in the data stream. In other words, the algorithm will be asked to answer queries of the form, "How many times did x occur in the stream?" and the goal will be to output an approximately correct answer with probability $1-\delta$. In this section we will present two different algorithms for this task. The algorithms have different benefits and drawbacks. The first algorithm has smaller space complexity and only suffers from one-sided error, i.e. it can overestimate the number of occurrences of x but it never underestimates. The second algorithm requires more space and suffers from two-sided error, but it satisfies a significantly stronger approximate-correctness property.

Algorithm 3 Count-Min Sketch

Given positive integers $B, t \ldots$
Sample $h_1, \ldots, h_t : [m] \to [B]$ independently from a 2-universal hash family.
Initialize a two-dimensional array C of dimensions $B \times t$, setting $C[k, \ell] = 0$ for each k, ℓ.
for each $i \in [n]$ **do**
    Observe a_i.
    for each $\ell \in [t]$ **do**
        Compute $k = h_\ell(a_i)$.
        Increment $C[k, \ell]$ by 1.
    end for
end for
When queried about frequency of token x, return $\min_{\ell \in [t]}\{C[h_\ell(x), \ell]\}$.

The first algorithm we'll analyze, called the Count-Min Sketch, is based on a hashing scheme presented in Algorithm 3. The idea behind the algorithm is simple: we choose t independent random hash functions $h_1, \ldots, h_t$, with range $[B]$ for some moderately large B, and for each "hash bucket" $k \in [B]$ we count how many elements of the stream are hashed to k by each of the t functions. If h is a hash function and x is a token appearing r times in the stream, then the counter for bucket $h(x)$ will reach a value which is at least r. To the extent that the counter exceeds r, the difference is due to hash collisions — other elements of the stream that hash to the same bucket as x. For large B, this will typically be only a small fraction of the stream. By repeating this counting procedure in parallel using t different hash functions, we minimize the probability of getting an anomalously large number of hash collisions.

Lemma 3.3. *The CountMin sketch uses space $s = O(Bt \log(mn))$ and satisfies the following guarantee for every $x \in [m]$: if the true frequency of x in the stream is denoted by f_x, the sketch's estimate $\hat{f}_x$ satisfies $f_x \leq \hat{f}_x$ with probability 1 and $\hat{f}_x \leq f_x + \frac{2n}{B}$ with probability at least $1 - 2^{-t}$.*

Proof. The space complexity bound follows from the observation that the algorithm only needs to store an array of dimensions $B \times t$, with each element of the array being an integer in the range $0, 1, \ldots, n$, plus descriptions of t hash functions each requiring space $O(\log m)$.

For each $\ell \in [t]$, the counter $C[h_\ell(x), \ell]$ is incremented each time x appears in the stream — f_x times in total — and it is also incremented each time another token $y \neq x$ appears in the stream and satisfies $h_\ell(y) = h_\ell(x)$. There are $n - f_x$ tokens other than x in the stream, and for each of them the probability that $h_\ell(y) = h_\ell(x)$ is $1/B$, so by linearity of expectation we have $\mathbb{E}[C[h_\ell(x), \ell] - f_x] = (n - f_x)/B$. Then, by Markov's Inequality,

$$\Pr\left(C[h_\ell(x), \ell] - f_x > \tfrac{2n}{B}\right) \leq \tfrac{1}{2}.$$

Since the hash function $\{h_1, \ldots, h_t\}$ are mutually independent,

$$\Pr\left(\forall \ell \in [t]\, C[h_\ell(x), \ell] - f_x > \tfrac{2n}{B}\right) \leq \left(\tfrac{1}{2}\right)^t,$$

and the lemma follows. □

Corollary 3.4. *For any $\varepsilon, \delta > 0$ the Count-Min Sketch with parameters $B = \lceil \frac{2}{\varepsilon} \rceil$ and $t = \lceil \log_2(1/\delta) \rceil$ achieves the following guarantee: for any token x, with probability at least $1 - \delta$ the estimated frequency of x differs from the true frequency by no more than εn. The space complexity of the sketch with these parameters is $O\left(\log(mn) \log(1/\delta)/\varepsilon\right)$.*

The second algorithm we'll analyze uses more space, namely $O\left(\log n \log(1/\delta)/\varepsilon^2\right)$, but achieves a stronger approximate-correctness guarantee: with probability at least $1 - \delta$, the estimate of f_x differs from the true value by at most $\varepsilon \|\mathbf{f}\|_2$. Here, $\mathbf{f}$ denotes the "frequency vector" of the stream, an m-dimensional vector whose x^{th} component f_x is the frequency of token x in the stream. Since the sum of frequencies of all tokens is n, we have $\mathbf{f}_1 = n$. Note that $\mathbf{f}_2 \leq \mathbf{f}_1$ for any vector $\mathbf{f}$, so the error bound of $\varepsilon \|\mathbf{f}\|_2$ is never worse than the εn error bound of the Count-Min Sketch. However, $\|\mathbf{f}\|_2$ can be much smaller than n; for example, when the tokens are uniformly distributed we have $\|\mathbf{f}\|_2 \approx \frac{n}{\min\{\sqrt{m}, \sqrt{n}\}}$.

Algorithm 4 Count Sketch

Given positive integers $B, t \ldots$
Sample $h_1, \ldots, h_t : [m] \to [B]$ independently from a 2-universal hash family.
Sample $g_1, \ldots, g_t : [m] \to \{\pm 1\}$ independently from a 2-universal hash family.
Initialize a two-dimensional array C of dimensions $B \times t$, setting $C[k, \ell] = 0$ for each k, ℓ.
for each $i \in [n]$ **do**
    Observe a_i.
    for each $\ell \in [t]$ **do**
        Compute $k = h_\ell(a_i)$.
        $C[k, \ell] \leftarrow C[k, \ell] + g_\ell(a_i)$.
    end for
end for
When queried about frequency of token x, return the median of the multiset $\{g_\ell(x) \cdot C[h_\ell(x), \ell]\}$.

The intuition for the Count Sketch is similar to that for the Count-Min Sketch with one important difference. As before, if x occurs f_x times in the stream, then with each occurrence we add $g_\ell(x)$ to $C[h_\ell(x), \ell]$, resulting in a total of $g_\ell(x) \cdot f_x$. Since $g_\ell(x)^2 = 1$, this means that the random variable $g_\ell(x) \cdot C[h_\ell(x), \ell]$ equals $f_x + Z$, where the random variable Z accounts for the "noise" due to other tokens $y \neq x$ that are hashed by h_ℓ to the same bucket as x, similarly to the analysis of the Count-Min Sketch. However, the key difference is that the noise variable Z in the Count Sketch is a sum of randomly-signed contributions from the various tokens that occupy the same hash bucket as x. In aggregate we can expect some of these noise terms to cancel each other out because they are oppositely signed. Hence, we might hope that the Count Sketch suffers from less additive error when estimating the frequency f_x. The following analysis substantiates that hope.

Lemma 3.5. *The Count Sketch uses space $s = O(Bt \log(mn))$ and satisfies the following guarantee for every $x \in [m]$: if the true frequency of x in the stream is denoted by f_x, the sketch's estimate $\hat{f}_x$ satisfies $|\hat{f}_x - f_x| \leq \sqrt{\frac{3}{B}} \|\mathbf{f}\|_2$ with probability at least $1 - e^{-t/18}$.*

Proof. Fix $x \in [m]$. For any $y \in [m]$ and $\ell \in [t]$ define random variables $X_{y\ell}$ and $Z_{y\ell}$ by

$$X_{y\ell} = \begin{cases} 1 & \text{if } h_\ell(y) = h_\ell(x) \\ 0 & \text{if } h_\ell(y) \neq h_\ell(x) \end{cases}$$
$$Z_{y\ell} = g_\ell(x) g_\ell(y) X_{y\ell} f_y.$$

In words, $X_{y\ell}$ equals 1 or 0 depending whether or not h_ℓ has a hash collision between y and x, and $Z_{y\ell}$ is a random variable representing the amount (positive or negative) that occurrences of token y in the stream contribute to the value of $g_\ell(x) \cdot C[h_\ell(x), \ell]$. To substantiate the latter interpretation, observe that

$$C[h_\ell(x), \ell] = \sum_{y=1}^{m} g_y(\ell) X_{y\ell} f_y$$

because token y occurs f_y times in the stream, and each of these occurrences contribute $g_y(\ell)$ to the counter $C[h_\ell(x), \ell]$ if and only if $X_{y\ell} = 1$, otherwise each occurrence of y in the stream has zero contribution to $C[h_\ell(x), \ell]$.

The random variable $Z_{x\ell}$ is deterministically equal to f_x because $g_\ell(x)^2 = 1$ and $X_{x\ell} = 1$. As for $Z_{y\ell}$ when $y \neq x$, we have

$$\mathbb{E}[Z_{y\ell}] = \mathbb{E}[g_\ell(x)g_\ell(y)X_{y\ell}f_y] = \mathbb{E}[g_\ell(x)] \cdot \mathbb{E}[g_\ell(y)] \cdot \mathbb{E}[X_{y\ell}] \cdot f_y = 0, \tag{14}$$

where we have used the fact that $g_\ell(x)$, $g_\ell(y)$, and $X_{y\ell}$ are mutually independent, and that $\mathbb{E}[g_\ell(x)] = \mathbb{E}[g_\ell(y)] = 0$. To verify the mutual independence, observe that $X_{y\ell}$ depends only on the hash function h_ℓ which is independent of g_ℓ, and the values $g_\ell(x), g_\ell(y)$ are independent of one another by the pairwise-independence property of g_ℓ.

Using linearity of expectation we have

$$\mathbb{E}[g_\ell(x) \cdot C[h_\ell(x), \ell]] = \sum_{y=1}^{m} \mathbb{E}[Z_{y\ell}] = f_x + \sum_{y \neq x} \mathbb{E}[Z_{y\ell}] = f_x. \tag{15}$$

To continue with the analysis of the Count Sketch, the next step is to analyze the variance of $g_\ell(x) \cdot C[h_\ell(x), \ell]$ and apply Chebyshev's Inequality. We have

$$\begin{aligned}
\mathrm{Var}[g_\ell(x) \cdot C[h_\ell(x), \ell]] &= \mathrm{Var}[f_x + \sum_{y \neq x} Z_{y\ell}] = \mathrm{Var}[\sum_{y \neq x} Z_{y\ell}] \\
&= \mathbb{E}\left[\left(\sum_{y \neq x} Z_{y\ell}\right)^2\right] \\
&= \sum_{y \neq x} \sum_{w \neq x} \mathbb{E}[Z_{y\ell}Z_{w\ell}] = \sum_{y \neq x} \mathbb{E}[Z_{y\ell}^2] + \sum_{y \neq x} \sum_{w \notin \{x,y\}} \mathbb{E}[Z_{y\ell}Z_{w\ell}].
\end{aligned}$$

Now,

$$\mathbb{E}[Z_{y\ell}^2] = \mathbb{E}[X_{y\ell}^2 f_y^2] = \mathbb{E}[X_{y\ell} f_y^2] = \tfrac{1}{B} f_y^2,$$

since $X_{y\ell} = 1$ with probability $\frac{1}{B}$ and $X_{y\ell} = 0$ otherwise. (Here we have used the fact that h_ℓ is drawn from a 2-universal hash family, so for any $y \neq x$ the probability of $h_\ell(y) = h_\ell(x)$ is $1/B$.) Furthermore, if $w \notin \{x, y\}$ then

$$\mathbb{E}[Z_{y\ell}Z_{w\ell}] = \mathbb{E}[g_\ell(y)g_\ell(w)X_{y\ell}X_{w\ell}f_y f_w] = \mathbb{E}[g_\ell(y)] \cdot \mathbb{E}[g_\ell(w)] \cdot \mathbb{E}[X_{y\ell}X_{w\ell}] \cdot f_y f_w = 0,$$

where we have again used the mutual independence of the random variables $g_\ell(y), g_\ell(w)$, and $X_{y\ell}X_{w\ell}$. (Note that $X_{y\ell}$ and $X_{w\ell}$ may be correlated with one another, we only need to use the fact that their product is independent of $g_\ell(y)$ and $g_\ell(w)$, which holds because $X_{y\ell}X_{w\ell}$ depends only on the hash function h_ℓ, which is independent of g_ℓ.) Substituting the calculated values of $\mathbb{E}[Z_{y\ell}^2]$ and $\mathbb{E}[Z_{y\ell}Z_{w\ell}]$ into the variance calculation, we find that

$$\mathrm{Var}[g_\ell(x) \cdot C[h_\ell(x), \ell]] = \frac{1}{B} \sum_{y \neq x} f_y^2 \leq \frac{1}{B} \|\mathbf{f}\|_2^2.$$

By Chebyshev's Inequality,

$$\Pr\left(|g_\ell(x) \cdot C[h_\ell(x), \ell] - f_x| \geq \sqrt{\tfrac{3}{B}}\,\|\mathbf{f}\|_2\right) \leq \frac{\mathrm{Var}[g_\ell(x) \cdot C[h_\ell(x), \ell]]}{\frac{3}{B}\|\mathbf{f}\|_2^2} = \frac{1}{3}. \tag{16}$$

We have shown that each of the individual estimates $g_\ell(x) \cdot C[h_\ell(x), \ell]$ has probability at most $\frac{1}{3}$ of differing from the target value f_x by more than $\sqrt{3/B} \cdot \|\mathbf{f}\|_2$. There are t such estimates, one for each $\ell \in [t]$, and they are independent random variables. In order for their *median* to be less than $f_x - \sqrt{3/B} \cdot \|\mathbf{f}\|_2$ or greater than $f_x + \sqrt{3/B} \cdot \|\mathbf{f}\|_2$, at least $t/2$ of the estimates must differ from f_x by more than $\sqrt{3/B} \cdot \|\mathbf{f}\|_2$. To finish up, we use the Hoeffding Bound to show that the probability of this happening is less than $e^{-t/18}$. In more detail, let W_ℓ be a random variable which equals 1 if $|g_\ell(x) \cdot C[h_\ell(x), \ell] - f_x| \geq \sqrt{3/B} \cdot \|\mathbf{f}\|_2$, otherwise $W_\ell = 0$. Inequality (16) says that $\mathbb{E}[W_\ell] \leq \frac{1}{3}$. Since the random variables $\{W_\ell : \ell \in [t]\}$ are mutually independent, Hoeffding's Inequality says that

$$\Pr\left(W_1 + \cdots + W_t \geq \tfrac{t}{2}\right) = \Pr\left(W_1 + \cdots + W_t \geq \mathbb{E}[W_1 + \cdots + W_t] + \tfrac{t}{6}\right) \leq e^{-2(t/6)^2/t} = e^{-t/18}.$$

□

Corollary 3.6. *For any $\varepsilon, \delta > 0$ the Count-Min Sketch with parameters $B = \lceil \frac{3}{\varepsilon^2} \rceil$ and $t = \lceil 18 \ln(1/\delta) \rceil$ achieves the following guarantee: for any token x, with probability at least $1 - \delta$ the estimated frequency of x differs from the true frequency by no more than $\varepsilon\|\mathbf{f}\|_2$. The space complexity of the sketch with these parameters is $O\left(\log(mn)\log(1/\delta)/\varepsilon^2\right)$.*

3.4 Quantile estimation

The last streaming algorithm we'll present in these notes is applicable when the tokens in the stream come from an ordered set. As in previous sections, we'll assume the tokens come from the set $[m] = \{1, 2, \ldots, m\}$, ordered by the $<$ relation. For convenience, we'll assume in this section that a single token can be stored in $O(1)$ space, i.e. that a single memory location can store $\log(m)$ bits. All of the algorithms we present here can be applied even when this assumption is violated; the space complexity bounds presented here would just need to be scaled by the amount of space required to store one token from the stream.

For a stream $a_1, \ldots, a_n$ and a token $a \in [m]$, we'll say the quantile of token a relative to the stream is

$$q(a) = \frac{\#\{i \mid a_i \leq a\}}{n}$$

and we'll say that $\hat{q}(a)$ is an ε-approximate quantile of a if $|\hat{q}(a) - q(a)| \leq \varepsilon$. Algorithms for *quantile estimation* maintain a sketch that enables them to estimate an ε-approximate quantile of any element.

In this section we'll present a simple algorithm for quantile estimation using $s = O\left(\frac{1}{\varepsilon^2} + \log n\right)$ bits of space. The design of the algorithm illustrates an important technique called *reservoir sampling* for maintaining a random sample of elements of a data stream. To analyze the algorithm we'll introduce and apply the Glivenko-Cantelli Theorem, an important theorem in statistics about estimating a univariate distribution from random samples.

3.4.1 Reservoir sampling

One of the most basic tasks in data streaming is downsampling: for a specific integer $t > 0$, draw t tokens uniformly at random from the multiset of n tokens in the stream. To state the goal of downsampling more precisely, the algorithm should output a sequence of t tokens, and the distribution of its output as an unordered multiset (i.e., ignoring the ordering of tokens in the output) should match the output distribution of a hypothetical algorithm that stores the entire stream, samples t distinct indices $1 \le i(1) < i(2) < \cdots < i(t) \le n$ uniformly at random from among all $\binom{n}{t}$ such t-tuples of indices, and outputs the multiset $\{a_{i(1)}, a_{i(2)}, \ldots, a_{i(t)}\}$.

Reservoir sampling is a simple and widely used downsampling procedure that works by maintaining a buffer $\mathbf{b} = (b_1, \ldots, b_t)$ of size t containing the random samples, and a counter that keeps track of the number of stream tokens seen so far. After an initialization phase that fills the buffer with the first t tokens of the stream, the token observed at time $s > t$ is discarded with probability $1 - \frac{t}{s}$, and otherwise a random element of the buffer is overwritten with the new token.

Algorithm 5 Reservoir sampling

```
Given positive integer t
for each s ∈ [n] do
    Observe a_s.
    if s ≤ t then
        Set b_s = a_s.
    else
        With probability t/s, sample index i ∈ [t] uniformly at random and set b_i = a_i.
    end if
end for
Output (b_1, ..., b_t).
```

Proposition 3.7. *When Algorithm 5 is used to process a stream of $n \ge t$ tokens, the indices of the t tokens it outputs constitute a uniformly random t-element subset of $[n]$.*

Proof. We use induction on n. In the base case $n = t$ the algorithm is guaranteed to output tokens $\{a_i \mid i \in [t]\}$, and $[t]$ is the only t-element subset of $[t]$.

When $n > t$, consider any t-tuple of distinct indices $1 \le i(1) < i(2) < \cdots < i(t) \le n$. If $i(t) < n$, then by the induction hypothesis the probability that the buffer stores elements $a_{i(1)}, \ldots, a_{i(t)}$ at the *start* of the final loop iteration is $1 \Big/ \binom{n-1}{t}$. With probability $1 - \frac{t}{n}$ the final loop iteration does not change the buffer, so the probability of outputting $\{a_{i(1)}, \ldots, a_{i(t)}\}$ is

$$\frac{1 - t/n}{\binom{n-1}{t}} = \frac{n-t}{n} \cdot \frac{t!(n-1-t)!}{(n-1)!} = \frac{t!(n-t)!}{n!} = \frac{1}{\binom{n}{t}}.$$

On the other hand, if $i(t) = n$, then the algorithm outputs $\{a_{i(1)}, \ldots, a_{i(t)}\}$ if and only if its buffer contents at the start of the final loop iteration are indexed by t-tuple $1 \le j(1) < j(2) < \cdots < j(t) \le n - 1$ such that the set $J_t = \{j(1), \ldots, j(t)\}$ contains $I_{t-1} = \{i(1), \ldots, i(t-1)\}$ as a subset. The number of t-element sets $J_t \subseteq [n]$ containing I_{t-1} as a subset is $n - t$, because $J_t \setminus I_{t-1}$ can be

any of the $n-t$ elements of $[n-1] \setminus I_{t-1}$. For each such set J_t, the probability of overwriting the unique element of $J_t \setminus I_{t-1}$ in the final iteration is $\frac{t}{n} \cdot \frac{1}{t} = \frac{1}{n}$. Since there are $n-t$ potential sets J_t, each having $1 \Big/ \binom{n-1}{t}$ of being in the buffer at the start of the final loop iteration, and overwriting $J_t \setminus I_{t-1}$ with $i(t) = n$ in the final iteration has probability $1/n$, we find that the probability of outputting $\{a_{i(1)}, a_{i(2)}, \ldots, a_{i(t)}\}$ is

$$(n-t) \cdot \frac{1}{\binom{n-1}{t}} \cdot \frac{1}{n} = \frac{n-t}{n} \cdot \frac{t!(n-1-t)!}{(n-1)!} = \frac{t!(n-t)!}{n!} = \frac{1}{\binom{n}{t}}.$$

□

3.4.2 Quantile estimation via reservoir sampling

Reservoir sampling leads to a very natural idea for quantile estimation: if the buffer $(b_1, \ldots, b_t)$ is a uniformly random t-element subset of the full stream, then it should constitute a representative sample of the full stream. If so, a good way to estimate the quantile of any token $a \in [m]$ with respect to the full stream is to calculate its quantile with respect to the tokens contained in the buffer.

The analysis of the quantile estimation algorithm will go more smoothly if we make a small modification to it. Rather than maintaining one reservoir sample of size t, our algorithm will maintain t independent reservoir samples, each of size 1. Reservoir sampling with $t = 1$ requires $O(1)$ space for the buffer and $O(\log n)$ space for the counter, so t independent reservoir sampling algorithms with a shared counter use $O(t + \log n)$ space.

Given t independent samples $b_1, \ldots, b_t$, each uniformly distributed over the multiset of tokens in the stream, the quantile estimate for any token $a \in [m]$ is

$$\hat{q}(a) = \frac{\#\{j \mid b_j \le a\}}{t}. \tag{17}$$

Proposition 3.8. *If $b_1, \ldots, b_t$ are independent random samples from the stream $a_1, \ldots, a_n$, each uniformly distributed over its n tokens, then for any $a \in [m]$ and $\varepsilon > 0$, the quantile estimate $\hat{q}(a)$ computed by Equation (17) satisfies $|\hat{q}(a) - q(a)| \le \varepsilon$ with probability at least $1 - 2e^{-2\varepsilon^2 t}$.*

Proof. We can bound the probability that the estimate differs from $q(a)$ by more than ε using Hoeffding's Inequality. For $1 \le j \le t$, let $X_j = 1$ if $b_j \le a$ and $X_j = 0$ otherwise. The random variables $\{X_j\}$ are independent and $\{0,1\}$-valued, and each has expected value $\mathbb{E}[X_j] = q(a)/n$. The event that $|\hat{q}(a) - q(a)| > \varepsilon$ is equivalent to the event that the sum $X = X_1 + \cdots + X_t$ satisfies $|X - \mathbb{E}[X]| > \varepsilon t$. Hence,

$$\Pr\left(|\hat{q}(a) - q(a)| > \varepsilon\right) = \Pr\left(|X - \mathbb{E}[X]| > \varepsilon t\right) < 2\exp\left(-\frac{2\varepsilon^2 t^2}{t}\right) = 2e^{-2\varepsilon^2 t}.$$

□

As a corollary, if we want each quantile query to have an (ε, δ)-PAC answer, then $t \ge \frac{1}{2}\varepsilon^{-2}\ln(2/\delta)$ independent uniformly random samples are sufficient.

3.4.3 Uniformly accurate quantile sketches via Glivenko-Cantelli

One can interpret the buffer of samples $(b_1, \ldots, b_t)$ used in Section 3.4.2 as a sketch that supports answering quantile queries via the formula (17). In light of that interpretation, Proposition 3.8 says that for any *particular* quantile query $q(a)$, the response $\hat{q}(a)$ is (ε, δ)-PAC when $t \geq \frac{1}{2}\varepsilon^{-2} \ln(2/\delta)$. However, we can ask for more: the sketch $(b_1, \ldots, b_t)$ is *uniformly ε-accurate for quantile queries* if $|\hat{q} - q|_\infty \leq \varepsilon$. In other words, a uniformly ε-accurate sketch supplies quantile estimates $\hat{q}$ that satisfy $|\hat{q}(a) - q(a)| \leq \varepsilon$ *simultaneously* for all $a \in [m]$.

Let us say an algorithm for quantile sketching is *uniformly (ε, δ)-PAC* if, for all input streams, the probability that the algorithm produces a uniformly ε-accurate sketch is at least $1 - \delta$.

A simple application of the union bound and Proposition 3.8 leads to the conclusion that $t \geq \frac{1}{2}\varepsilon^{-2} \ln(2m/\delta)$ independent uniformly random samples suffice for a uniformly (ε, δ)-PAC quantile sketch. That's because when t satisfies the specified lower bound, for any $a \in [m]$ an application of Proposition 3.8 ensures that

$$\Pr\left(|\hat{q}(a) - q(a)| > \varepsilon\right) \leq \frac{\delta}{m}.$$

Summing over $a \in [m]$ and applying the union bound, we find that the quantile sketch obtained from $(b_1, \ldots, b_t)$ is uniformly (ε, δ)-PAC. However, setting $t \geq \frac{1}{2}\varepsilon^{-2} \ln(2m/\delta)$ results in a sketch whose space complexity is logarithmic in m. A more careful analysis will justify using a number of samples with *no dependence on m at all.* The key is the following inequality for independent samples drawn from any univariate distribution.

Lemma 3.9. *Let $b_1, \ldots, b_t$ be t independent, identically distributed random numbers each with cumulative distribution function F. The function $\hat{q}$ defined by formula (17) satisfies*

$$\forall \varepsilon > 0 \quad \Pr\left(|\hat{q} - F|_\infty > \varepsilon\right) < \frac{4}{\varepsilon} e^{-\varepsilon^2 t/2}. \tag{18}$$

Proof. Consider any $\varepsilon > 0$, and let $k = \lceil 2/\varepsilon \rceil$. For $1 \leq i \leq k$ let

$$a^{(i)} = \sup\left\{a \mid F(a) < \tfrac{i}{k}\right\}.$$

By the definition of $a^{(i)}$, we have

$$\forall a < a^{(i)} \quad F(a) < \tfrac{i}{k}. \tag{19}$$

Furthermore, since cumulative distribution functions are right-continuous, we have

$$F(a^{(i)}) = \inf\{F(a) \mid a > a^{(i)}\} \geq \frac{i}{k}. \tag{20}$$

For each $a \in \mathbb{R}$ define $\hat{q}(a)$ as in Equation (17) and define

$$\check{q}(a) = \frac{\#\{j \mid b_j < a\}}{t}. \tag{21}$$

Let $F(a^{(i)}-)$ denotes $\sup\{F(a) \mid a < a^{(i)}\}$, the probability of sampling a value strictly less than $a^{(i)}$ under the common distribution of $b_1, \ldots, b_t$. We claim that whenever $\|\hat{q} - F\|_\infty > \varepsilon$ there exists some $i \in [k-1]$ such that either $F(a^{(i)}) - \hat{q}(a^{(i)}) > \varepsilon/2$ or $\check{q}(a^{(i)}) - F(a^{(i)}-) > \varepsilon/2$. To prove the claim, we suppose $|\hat{q}(a) - F(a)| > \varepsilon$ for some $a \in \mathbb{R}$ and perform the following case analysis.

1. If $F(a) - \hat{q}(a) > \varepsilon$ then let i/k denote the greatest multiple of $1/k$ less than $F(a)$. Note that $i \leq k-1$ because $\frac{i}{k} < F(a) \leq 1$. We have $F(a^{(i)}) \geq \frac{i}{k}$ by (20) whereas $\hat{q}(a^{(i)}) \leq \hat{q}(a)$ by monotonicity of $\hat{q}$. Hence, subtracting the two inequalities,

$$F(a^{(i)}) - \hat{q}(a^{(i)}) \;\geq\; \tfrac{i}{k} - \hat{q}(a) \;\geq\; F(a) - \tfrac{1}{k} - \hat{q}(a) \;>\; \varepsilon - \tfrac{1}{k} \;>\; \tfrac{\varepsilon}{2}.$$

2. If $\hat{q}(a) - F(a) > \varepsilon$ then let i/k denote the least multiple of $1/k$ greater than $F(a)$. Note that $i \leq k-1$ because

$$\frac{i+1}{k} \leq F(a) + \frac{2}{k} \leq F(a) + \varepsilon < \hat{q}(a) \leq 1.$$

   Since $F(a) < \frac{i}{k}$ we have $a < a^{(i)}$ and hence $\hat{q}(a) \leq \check{q}(a^{(i)})$. Then,

$$\check{q}(a^{(i)}) - F(a^{(i)}-) \;\geq\; \hat{q}(a) - F(a^{(i)}-) \;\geq\; \hat{q}(a) - \tfrac{i}{k} \;\geq\; \hat{q}(a) - F(a) - \tfrac{1}{k} \;>\; \varepsilon - \tfrac{1}{k} \;>\; \tfrac{\varepsilon}{2}.$$

For each $i \in [k]$ an application of Hoeffding's inequality as in Proposition 3.8 ensures that

$$\Pr\left(F(a^{(i)}) - \hat{q}(a^{(i)}) > \varepsilon/2\right) \qquad < e^{-\varepsilon^2 t/2}$$

and

$$\Pr\left(\check{q}(a^{(i)}) - F(a^{(i)}-) > \varepsilon/2\right) \qquad < e^{-\varepsilon^2 t/2}$$

By the union bound,

$$\Pr\left(\exists i \in [k-1]\ F(a^{(i)}) - \hat{q}(a^{(i)}) > \varepsilon/2 \text{ or } \check{q}(a^{(i)}) - F(a^{(i)}-) > \varepsilon/2\right) < 2(k-1)e^{-\varepsilon^2 t/2} < \frac{4}{\varepsilon} e^{-\varepsilon^2 t/2} \quad (22)$$

where the second inequality follows because $k - 1 < 2/\varepsilon$, by our choice of k. □

Lemma 3.9 has two corollaries that are worthy of note. The first is an upper bound with on the number of samples needed for uniformly (ε, δ)-PAC quantile sketching. As promised, the bound has no dependence on m, the size of the set from which tokens are drawn.

Corollary 3.10. *If $t \geq 2\varepsilon^{-2} \ln\left(\frac{4}{\varepsilon\delta}\right)$ then the quantile sketch obtained from t independent uniformly-distributed stream tokens is uniformly (ε, δ)-PAC.*

The second corollary of Lemma 3.9 is the Glivenko-Cantelli Theorem, an important theorem in statistics asserting that the empirical distribution of n independent, identically distributed random variables converges uniformly to the distributon from which the variables were drawn, as n tends to infinity.

Theorem 3.11. *If $X_1, X_2, \ldots$ is an infinite sequence of independent, identically distributed random variables, each with cumulative distribution function F, and if F_t denotes the empirical cumulative distribution function*

$$F_t(a) = \frac{\sharp\{i \mid X_i \leq a \text{ and } 1 \leq i \leq t\}}{t}$$

then $\lim_{t\to\infty} \|F_t - F\|_\infty = 0$ almost surely.

Proof. The function F_t is identical to the quantile estimate $\hat{q}$ determined by the buffer $(b_1, \ldots, b_t) = (X_1, \ldots, X_t)$. Hence, for any $\varepsilon > 0$ we can apply Lemma 3.9 and linearity of expectation to conclude that the expected number of t such that $\|F_t - F\|_\infty > \varepsilon$ is less than

$$\frac{4}{\varepsilon} \sum_{t=1}^{\infty} e^{-\varepsilon^2 t/2} = \frac{4}{\varepsilon} \cdot \frac{1}{1 - e^{-\varepsilon^2/2}} < \infty.$$

By the Borel-Cantelli Lemma, the number of t such that $\|F_t - F\|_\infty > \varepsilon$ is almost surely finite. In other words,

$$\Pr\left(\limsup_{t \to \infty} \|F_t - F\|_\infty > \varepsilon\right) = 0.$$

Since the above probability equals zero for every ε, it follows that $\lim_{t\to\infty} \|F_t - F\|_\infty = 0$ almost surely. □

One concluding remark is that the bound in Lemma 3.9 can be tightened further. The sharp bound is called the Dvoretzky-Kiefer-Wolfowitz Inequality (or DKW Inequality) and takes the following form.

$$\forall \varepsilon > 0 \quad \Pr\left(\|\hat{q} - F\|_\infty > \varepsilon\right) < 2e^{-2\varepsilon^2 t}. \tag{23}$$

The inequality is quite remarkable, because the right-hand side is the same as the upper bound on $\Pr\left(|\hat{q}(a) - F(a)| > \varepsilon\right)$ for a *single* value of a derived using Hoeffding's Inequality in Proposition 3.8. The DKW Inequality pertains of to the union of the events $\{|\hat{q}(a) - F(a)| > \varepsilon\}$ for *uncountably many* values of a, yet somehow taking the union of all these events comes at no cost in terms of the probability bound.

The DKW inequality justifies that the reservoir-sampling-based quantile sketch is uniformly (ε, δ)-PAC as long as $t \geq \frac{1}{2}\varepsilon^{-2} \ln(2/\delta)$. The sketching algorithm runs in

$$s = O(\varepsilon^{-2} \log(1/\delta) + \log_m(n))$$

bits of space, with the $\log_m(n)$ bits being needed for the counter in the reservoir sampling algorithm. A different quantile estimation algorithm, due to Greenwald and Khanna, runs in $O(\varepsilon^{-1} \log(\varepsilon n))$ space, improving the dependence on $1/\varepsilon$ from quadratic to quasi-linear at the cost of slightly worse dependence on $\log(n)$.