since $\mathbf{Var}[f(X_i)] \leq \mathbf{E}[(f(X_i))^2] \leq 1$. We therefore find $\Pr(|Y - \bar{f}| \geq \varepsilon) \leq \delta$ when $m = 1/\delta\varepsilon^2$. (In fact, one can prove that $\mathbf{Var}[f(X_i)] \leq 1/4$, giving a slightly better bound; this is left as Exercise 13.4.)

Using pairwise independent samples requires more samples: $\Theta(1/\delta\varepsilon^2)$ instead of the $\Theta(\ln(1/\delta)/\varepsilon^2)$ samples when they are independent. But recall from Section 13.1.3 that we can obtain up to 2^n pairwise independent samples with just $2n$ uniform independent bits. Hence, as long as $1/\delta\varepsilon^2 < 2^n$, just $2n$ random bits suffice; this is much less than the number required when using completely independent samples. Usually ε and δ are fixed constants independent of n, and this type of estimation is quite efficient in terms of both the number of random bits used and the computational cost.

13.3. Families of Universal Hash Functions

Up to this point, when studying hash functions we modeled them as being completely random in the sense that, for any collection of items $x_1, x_2, \ldots, x_k$, the hash values $h(x_1), h(x_2), \ldots, h(x_k)$ were considered uniform and independent over the range of the hash function. This was the framework we used to analyze hashing as a balls-and-bins problem in Chapter 5. The assumption of a completely random hash function simplifies the analysis for a theoretical study of hashing. In practice, however, completely random hash functions are too expensive to compute and store, so the model does not fully reflect reality.

Two approaches are commonly used to implement practical hash functions. In many cases, heuristic or ad hoc functions designed to appear random are used. Although these functions may work suitably for some applications, they generally do not have any associated provable guarantees, making their use potentially risky. Another approach is to use hash functions for which there are some provable guarantees. We trade away the strong statements one can make about completely random hash functions for weaker statements with hash functions that are efficient to store and compute.

We consider one of the computationally simplest classes of hash functions that provide useful provable performance guarantees: universal families of hash functions. These functions are widely used in practice.

Definition 13.2: *Let U be a universe with $|U| \geq n$ and let $V = \{0, 1, \ldots, n-1\}$. A family of hash functions $\mathcal{H}$ from U to V is said to be* k-universal *if, for any elements $x_1, x_2, \ldots, x_k$ and for a hash function h chosen uniformly at random from $\mathcal{H}$, we have*

$$\Pr(h(x_1) = h(x_2) = \cdots = h(x_k)) \leq \frac{1}{n^{k-1}}.$$

A family of hash functions $\mathcal{H}$ from U to V is said to be strongly k-universal *if, for any elements $x_1, x_2, \ldots, x_k$, any values $y_1, y_2, \ldots, y_k \in \{0, 1, \ldots, n-1\}$, and a hash function h chosen uniformly at random from $\mathcal{H}$, we have*

$$\Pr((h(x_1) = y_1) \cap (h(x_2) = y_2) \cap \cdots \cap (h(x_k) = y_k)) = \frac{1}{n^k}.$$

We will primarily be interested in 2-universal and strongly 2-universal families of hash functions. When we choose a hash function from a family of 2-universal hash functions, the probability that any two elements x_1 and x_2 have the same hash value is at most $1/n$. In this respect, a hash function chosen from a 2-universal family acts like a random hash function. It does not follow, however, that for 2-universal families the probability of any three values x_1, x_2, and x_3 having the same hash value is at most $1/n^2$, as would be the case if the hash values of x_1, x_2, and x_3 were mutually independent.

When a family is strongly 2-universal and we choose a hash function from that family, the values $h(x_1)$ and $h(x_2)$ are pairwise independent, since the probability that they take on any specific pair of values is $1/n^2$. Because of this, hash functions chosen from a strongly 2-universal family are also known as *pairwise independent* hash functions. More generally, if a family is strongly k-universal and we choose a hash function from that family, then the values $h(x_1), h(x_2), \ldots, h(x_k)$ are k-wise independent. Notice that a strongly k-universal hash function is also k-universal.

To gain some insight into the behavior of universal families of hash functions, let us revisit a problem we considered in the balls-and-bins framework of Chapter 5. We saw in Section 5.2 that, when n items are hashed into n bins by a completely random hash function, the maximum load is $\Theta(\log n / \log \log n)$ with high probability. We now consider what bounds can be obtained on the maximum load when n items are hashed into n bins using a hash function chosen from a 2-universal family.

First, consider the more general case where we have m items labeled $x_1, x_2, \ldots, x_m$. For $1 \leq i < j \leq m$, let $X_{ij} = 1$ if items x_i and x_j land in the same bin. Let $X = \sum_{1 \leq i < j \leq m} X_{ij}$ be the total number of collisions between pairs of items. By the linearity of expectations,

$$\mathbf{E}[X] = \mathbf{E}\left[\sum_{1 \leq i < j \leq m} X_{ij}\right] = \sum_{1 \leq i < j \leq m} \mathbf{E}[X_{ij}].$$

Since our hash function is chosen from a 2-universal family, it follows that

$$\mathbf{E}[X_{ij}] = \Pr(h(x_i) = h(x_j)) \leq \frac{1}{n}$$

and hence

$$\mathbf{E}[X] \leq \binom{m}{2}\frac{1}{n} < \frac{m^2}{2n}. \tag{13.1}$$

Markov's inequality then yields

$$\Pr\left(X \geq \frac{m^2}{n}\right) \leq \Pr(X \geq 2\mathbf{E}[X]) \leq \frac{1}{2}.$$

If we now suppose that the maximum number of items in a bin is Y, then the number of collisions X must be at least $\binom{Y}{2}$. Therefore,

$$\Pr\left(\binom{Y}{2} \geq \frac{m^2}{n}\right) \leq \Pr\left(X \geq \frac{m^2}{n}\right) \leq \frac{1}{2},$$

which implies that

$$\Pr\left(Y \geq m\sqrt{2/n}\right) \leq \frac{1}{2}.$$

In particular, in the case where $m = n$, the maximum load is at most $\sqrt{2n}$ with probability at least $1/2$.

This result is much weaker than the one for perfectly random hash functions, but it is extremely general in that it holds for any 2-universal family of hash functions. The result will prove useful for designing perfect hash functions, as we describe in Section 13.3.3.

13.3.1. *Example: A 2-Universal Family of Hash Functions*

Let the universe U be the set $\{0, 1, 2, \ldots, m-1\}$ and let the range of our hash function be $V = \{0, 1, 2, \ldots, n-1\}$, with $m \geq n$. Consider the family of hash functions obtained by choosing a prime $p \geq m$, letting

$$h_{a,b}(x) = ((ax + b) \bmod p) \bmod n$$

and then taking the family

$$\mathcal{H} = \{h_{a,b} \mid 1 \leq a \leq p-1,\ 0 \leq b \leq p\}.$$

Notice that a cannot here take on the value 0.

Lemma 13.6: *$\mathcal{H}$ is 2-universal.*

Proof: We count the number of functions in $\mathcal{H}$ for which two distinct elements x_1 and x_2 from U collide.

First we note that, for any $x_1 \neq x_2$,

$$ax_1 + b \neq ax_2 + b \bmod p.$$

This follows because $ax_1 + b = ax_2 + b \bmod p$ implies that $a(x_1 - x_2) = 0 \bmod p$, yet here both a and $(x_1 - x_2)$ are nonzero modulo p.

In fact, for every pair of values (u, v) such that $u \neq v$ and $0 \leq u, v \leq p-1$, there exists exactly one pair of values (a, b) for which $ax_1 + b = u \bmod p$ and $ax_2 + b = v \bmod p$. This pair of equations has two unknowns, and its unique solution is given by:

$$a = \frac{v-u}{x_2 - x_1} \bmod p,$$
$$b = u - ax_1 \bmod p.$$

Since there is exactly one hash function for each pair (a, b), it follows that there is exactly one hash function in $\mathcal{H}$ for which

$$ax_1 + b = u \bmod p \quad \text{and} \quad ax_2 + b = v \bmod p.$$

Therefore, in order to bound the probability that $h_{a,b}(x_1) = h_{a,b}(x_2)$ when $h_{a,b}$ is chosen uniformly at random from $\mathcal{H}$, it suffices to count the number of pairs (u, v), $0 \leq u, v \leq p-1$, for which $u \neq v$ but $u = v \bmod n$. For each choice of u there are

at most $\lceil p/n \rceil - 1$ possible appropriate values for v, giving at most $p(\lceil p/n \rceil - 1) \leq p(p-1)/n$ pairs. Each pair corresponds to one of $p(p-1)$ hash functions, so

$$\Pr(h_{a,b}(x_1) = h_{a,b}(x_2)) \leq \frac{p(p-1)/n}{p(p-1)} = \frac{1}{n},$$

proving that $\mathcal{H}$ is 2-universal. ■

13.3.2. *Example: A Strongly 2-Universal Family of Hash Functions*

We can apply ideas similar to those used to construct the 2-universal family of hash functions in Lemma 13.6 to construct strongly 2-universal families of hash functions. To start, suppose that both our universe U and the range V of the hash function are $\{0, 1, 2, \ldots, p-1\}$ for some prime p. Now let

$$h_{a,b}(x) = (ax + b) \bmod p,$$

and consider the family

$$\mathcal{H} = \{h_{a,b} \mid 0 \leq a, b \leq p-1\}.$$

Notice that here a can take on the value 0, in contrast with the family of hash functions used in Lemma 13.6.

Lemma 13.7: *$\mathcal{H}$ is strongly 2-universal.*

Proof: This is entirely similar to the proof of Lemma 13.2. For any two elements x_1 and x_2 in U and any two values y_1 and y_2 in V, we need to show that

$$\Pr((h_{a,b}(x_1) = y_1) \cap (h_{a,b}(x_2) = y_2)) = \frac{1}{p^2}.$$

The condition that both $h_{a,b}(x_1) = y_1$ and $h_{a,b}(x_2) = y_2$ yields two equations modulo p with two unknowns, the values for a and b: $ax_1 + b = y_1 \bmod p$ and $ax_2 + b = y_2 \bmod p$. This system of two equations and two unknowns has just one solution:

$$a = \frac{y_2 - y_1}{x_2 - x_1} \bmod p,$$
$$b = y_1 - ax_1 \bmod p.$$

Hence only one choice of the pair (a, b) out of the p^2 possibilities results in x_1 and x_2 hashing to y_1 and y_2, proving that

$$\Pr((h_{a,b}(x_1) = y_1) \cap (h_{a,b}(x_2) = y_2)) = \frac{1}{p^2},$$

as required. ■

Although this gives a strongly 2-universal hash family, the restriction that the universe U and the range V be the same makes the result almost useless; usually we want to hash a large universe into a much smaller range. We can extend the construction in a natural way that allows much larger universes. Let $V = \{0, 1, 2, \ldots, p-1\}$, but now

let $U = \{0, 1, 2, \ldots, p^k - 1\}$ for some integer k and prime p. We can interpret an element u in the universe U as a vector $\bar{u} = (u_0, u_1, \ldots, u_{k-1})$, where $0 \leq u_i \leq p - 1$ for $0 \leq i \leq k - 1$ and where $\sum_{i=0}^{k-1} u_i p^i = u$. In fact, this gives a one-to-one mapping between vectors of this form and elements of U.

For any vector $\bar{a} = (a_0, a_1, \ldots, a_{k-1})$ with $0 \leq a_i \leq p - 1$, $0 \leq i \leq k - 1$, and for any value b with $0 \leq b \leq p - 1$, let

$$h_{\bar{a},b}(u) = \left(\sum_{i=0}^{k-1} a_i u_i + b \right) \bmod p,$$

and consider the family

$$\mathcal{H} = \{h_{\bar{a},b} \mid 0 \leq a_i, b \leq p - 1 \text{ for all } 0 \leq i \leq k - 1\}.$$

Lemma 13.8: *$\mathcal{H}$ is strongly 2-universal.*

Proof: We follow the proof of Lemma 13.7. For any two elements u_1 and u_2 with corresponding vectors $\bar{u}_i = (u_{i,0}, u_{i,1}, \ldots, u_{i,k-1})$ and for any two values y_1 and y_2 in V, we need to show that

$$\Pr((h_{\bar{a},b}(u_1) = y_1) \cap (h_{\bar{a},b}(u_2) = y_2)) = \frac{1}{p^2}.$$

Since u_1 and u_2 are different, they must differ in at least one coordinate. Without loss of generality let $u_{1,0} \neq u_{2,0}$. For any given values of $a_1, a_2, \ldots, a_{k-1}$, the condition that $h_{\bar{a},b}(u_1) = y_1$ and $h_{\bar{a},b}(u_2) = y_2$ is equivalent to:

$$a_0 u_{1,0} + b = \left(y_1 - \sum_{j=1}^{k-1} a_j u_{1,j} \right) \bmod p$$

$$a_0 u_{2,0} + b = \left(y_1 - \sum_{j=1}^{k-1} a_j u_{2,j} \right) \bmod p.$$

For any given values of $a_1, a_2, \ldots, a_{k-1}$, this gives a system with two equations and two unknowns (namely, a_0 and b), which – as in Lemma 13.8 – has exactly one solution. Hence, for every $a_1, a_2, \ldots, a_{k-1}$, only one choice of the pair (a_0, b) out of the p^2 possibilities results in u_1 and u_2 hashing to y_1 and y_2, proving that

$$\Pr((h_{\bar{a},b}(u_1) = y_1) \cap (h_{\bar{a},b}(u_2) = y_2)) = \frac{1}{p^2},$$

as required. ■

Although we have described both the 2-universal and the strongly 2-universal hash families in terms of arithmetic modulo a prime number, we could extend these techniques to work over general finite fields – in particular, fields with 2^n elements represented by sequences of n bits. The extension requires knowledge of finite fields, so we just sketch the result here. The setup and proof are exactly the same as for Lemma 13.8 except that, instead of working modulo p, we perform all arithmetic in a fixed finite field

with 2^n elements. We assume a fixed one-to-one mapping f from strings of n bits, which can also be thought of as numbers in $\{0, 1, \ldots, 2^n - 1\}$, to field elements. We let

$$h_{\vec{a},b}(u) = f^{-1}\left(\sum_{i=0}^{k-1} f(a_i) \cdot f(u_i) + f(b)\right),$$

where the a_i and b are chosen independently and uniformly over $\{0, 1, \ldots, 2^n - 1\}$ and where the addition and multiplication are performed over the field. This gives a strongly 2-universal hash function with a range of size 2^n.

13.3.3. *Application: Perfect Hashing*

Perfect hashing is an efficient data structure for storing a *static dictionary*. In a static dictionary, items are permanently stored in a table. Once the items are stored, the table is used only for search operations: a search for an item gives the location of the item in the table or returns that the item is not in the table.

Suppose that a set S of m items is hashed into a table of n bins, using a hash function from a 2-universal family and chain hashing. In chain hashing (see Section 5.5.1), items hashed to the same bin are kept in a linked list. The number of operations for looking up an item x is proportional to the number of items in x's bin. We have the following simple bound.

Lemma 13.9: *Assume that m elements are hashed into an n-bin chain hashing table by using a hash function h chosen uniformly at random from a 2-universal family. For an arbitrary element x, let X be the number of items at the bin $h(x)$. Then*

$$\mathbf{E}[X] \leq \begin{cases} m/n & \text{if } x \notin S, \\ 1 + (m-1)/n & \text{if } x \in S. \end{cases}$$

Proof: Let $X_i = 1$ if the ith element of S (under some arbitrary ordering) is in the same bin as x and 0 otherwise. Because the hash function is chosen from a 2-universal family, it follows that

$$\Pr(X_i = 1) = 1/n.$$

Then the first result follows from

$$\mathbf{E}[X] = \mathbf{E}\left[\sum_{i=1}^{m} X_i\right] = \sum_{i=1}^{m} \mathbf{E}[X_i] \leq \frac{m}{n},$$

where we have used the universality of the hash function to conclude that $\mathbf{E}[X_i] \leq 1/n$. Similarly, if x is an element of S then (without loss of generality) let it be the first element of S. Hence $X_1 = 1$, and again

$$\Pr(X_i = 1) = 1/n$$

when $i \neq 1$. Therefore,

$$\mathbf{E}[X] = \mathbf{E}\left[\sum_{i=1}^{m} X_i\right] = 1 + \sum_{i=2}^{m} \mathbf{E}[X_i] \leq 1 + \frac{m-1}{n}.$$ ■

Lemma 13.9 shows that the average performance of hashing when using a hash function from a 2-universal family is good, since the time to look through a bin of any item is bounded by a small number. For instance, if $m = n$ then, when searching the hash table for x, the expected number of items other than x that must be examined is at most 1. However, this does not give us a bound on the worst-case time of a lookup. Some bin may contain $\sqrt{n}$ elements or more, and a search for one of these elements requires a much longer lookup time.

This motivates the idea of *perfect hashing*. Given a set S, we would like to construct a hash table that gives excellent worst-case performance. Specifically, by perfect hashing we mean that only a constant number of operations are required to find an item in a hash table (or to determine that it isn't there).

We first show that perfect hashing is easy if we are given sufficient space for the hash table and a suitable 2-universal family of hash functions.

Lemma 13.10: *If $h \in \mathcal{H}$ is chosen uniformly at random from a 2-universal family of hash functions mapping the universe U to $[0, n-1]$ then, for any set $S \subset U$ of size m, the probability of h being perfect is at least $1/2$ when $n \geq m^2$.*

Proof: Let $s_1, s_2, \ldots, s_m$ be the m items of S. Let X_{ij} be 1 if the $h(s_i) = h(s_j)$ and 0 otherwise. Let $X = \sum_{1 \leq i < j \leq m} X_{ij}$. Then, as we saw in Eqn. (13.1), the expected number of collisions when using a 2-universal hash function is

$$\mathbf{E}[X] = \mathbf{E}\left[\sum_{1 \leq i < j \leq m} X_{ij}\right] = \sum_{1 \leq i < j \leq m} \mathbf{E}[X_{ij}] \leq \binom{m}{2}\frac{1}{n} < \frac{m^2}{2n}.$$

Markov's inequality then yields

$$\Pr\left(X \geq \frac{m^2}{n}\right) \leq \Pr(X \geq 2\mathbf{E}[X]) \leq \frac{1}{2}.$$

Hence, when $n \geq m^2$, we find $X < 1$ with probability at least $1/2$. This implies that a randomly chosen hash function is perfect with probability at least $1/2$. ■

To find a perfect hash function when $n \geq m^2$, we may simply try hash functions chosen uniformly at random from the 2-universal family until we find one with no collisions. This gives a Las Vegas algorithm. On average we need to try at most two hash functions.

We would like to have perfect hashing without requiring space for $\Omega(m^2)$ bins to store the set of m items. We can use a two-level scheme that accomplishes perfect hashing using only $O(m)$ bins. First, we hash the set into a hash table with m bins using a hash function from a 2-universal family. Some of these bins will have collisions. For each such bin, we provide a second hash function from an appropriate 2-universal family and an entirely separate second hash table. If the bin has $k > 1$ items in it then we use k^2 bins in the secondary hash table. We have already shown in Lemma 13.10 that with k^2 bins we can find a hash function from a 2-universal family that will give no collisions. It remains to show that, by carefully choosing the first hash function, we can guarantee that the total space used by the algorithm is only $O(m)$.

Theorem 13.11: *The two-level approach gives a perfect hashing scheme for m items using $O(m)$ bins.*

Proof: As we showed in Lemma 13.10, the number of collisions X in the first stage satisfies

$$\Pr\left(X \geq \frac{m^2}{n}\right) \leq \Pr(X \geq 2\mathbf{E}[X]) \leq \frac{1}{2}.$$

When $n = m$, this implies that the probability of having more than m collisions is at most $1/2$. Using the probabilistic method, there exists a choice of hash function from the 2-universal family in the first stage that gives at most m collisions. In fact, such a hash function can be found efficiently by trying hash functions chosen uniformly at random from the 2-universal family, giving a Las Vegas algorithm. We may therefore assume that we have found a hash function for the first stage that gives at most m collisions.

Let c_i be the number of items in the ith bin. Then there are $\binom{c_i}{2}$ collisions between items in the ith bin, so

$$\sum_{i=1}^{m} \binom{c_i}{2} \leq m.$$

For each bin with $c_i > 1$ items, we find a second hash function that gives no collisions using space c_i^2. Again, for each bin, this hash function can be found using a Las Vegas algorithm. The total number of bins used is then bounded above by

$$m + \sum_{i=1}^{m} c_i^2 \leq m + 2\sum_{i=1}^{m} \binom{c_i}{2} + \sum_{i=1}^{m} c_i \leq m + 2m + m = 4m.$$

Hence, the total number of bins used is only $O(m)$. ■

13.4. Application: Finding Heavy Hitters in Data Streams

A router forwards packets through a network. At the end of the day, a natural question for a network administrator to ask is whether the number of bytes traveling from a source s to a destination d that have passed through the router is larger than a predetermined threshold value. We call such a source–destination pair a *heavy hitter.*

When designing an algorithm for finding heavy hitters, we must keep in mind the restrictions of the router. Routers have very little memory and so cannot keep a count for each possible pair s and d, since there are simply too many such pairs. Also, routers must forward packets quickly, so the router must perform only a small number of computational operations for each packet. We present a randomized data structure that is appropriate even with these limitations. The data structure requires a *threshold* q; all source–destination pairs that are responsible for at least q total bytes are considered heavy hitters. Usually q is some fixed percentage, such as 1%, of the total expected daily traffic. At the end of the day, the data structure gives a list of possible heavy hitters. All true heavy hitters (responsible for at least q bytes) are listed, but some other

pairs may also appear in the list. Two other input constants, ε and δ, are used to control what extraneous pairs might be put in the list of heavy hitters. Suppose that Q represents the total number of bytes over the course of the day. Our data structure has the guarantee that any source–destination pair that constitutes less than $q - \varepsilon Q$ bytes of traffic is listed with probability at most δ. In other words, all heavy hitters are listed; all pairs that are sufficiently far from being a heavy hitter are listed with probability at most δ; pairs that are close to heavy hitters may or may not be listed.

This router example is typical of many situations where one wants to keep a succinct summary of a large data stream. In most *data stream models*, large amounts of data arrive sequentially in small blocks, and each block must be processed before the next block arrives. In the setting of network routers, each block is generally a packet. The amount of data being handled is often so large and the time between arrivals is so small that algorithms and data structures that use only a small amount of memory and computation per block are required.

We can use a variation of a Bloom filter, discussed in Section 5.5.3, to solve this problem. Unlike our solution there, which assumed the availability of completely random hash functions, here we obtain strong, provable bounds using only a family of 2-universal hash functions. This is important, because efficiency in the router setting demands the use of only very simple hash functions that are easy to compute, yet at the same time we want provable performance guarantees.

We refer to our data structure as a *count-min filter*. The count-min filter processes a sequential stream of pairs $X_1, X_2, \ldots$ of the form $X_t = (i_t, c_t)$, where i_t is an item and $c_t > 0$ is an integer count increment. In our routing setting, i_t would be the pair of source–destination addresses of a packet and c_t would be the number of bytes in the packet. Let

$$\text{Count}(i, T) = \sum_{t: i_t = i, 1 \le t \le T} c_t.$$

That is, $\text{Count}(i, T)$ is the total count associated with an item i up to time T. In the routing setting, $\text{Count}(i, T)$ would be the total number of bytes associated with packets with an address pair i up to time T. The count-min filter keeps a running approximation of $\text{Count}(i, T)$ for all items i and all times T in such a way that it can track heavy hitters.

A count-min filter consist of m counters. We assume henceforth that our counters have sufficiently many bits that we do not need to worry about overflow; in many practical situations, 32-bit counters will suffice and are convenient for implementation. A count-min filter uses k hash functions. We split the counters into k disjoint groups $G_1, G_2, \ldots, G_k$ of size m/k. For convenience, we assume in what follows that k divides m evenly. We label the counters by $C_{a,j}$, where $1 \le a \le k$ and $0 \le j \le m/k - 1$, so that $C_{a,j}$ corresponds to the jth counter in the ath group. That is, we can think of our counters as being organized in a 2-dimensional array, with m/k counters per row and k columns. Our hash functions should map items from the universe into counters, so we have hash functions H_a for $1 \le a \le k$, where $H_a : U \to [0, m/k - 1]$. That is, each of the k hash functions takes an item from the universe and maps it into a number $[0, m/k - 1]$. Equivalently, we can think of each hash function as taking an item

i and mapping it to the counter $C_{a,H_a(i)}$. The H_a should be chosen independently and uniformly at random from a 2-universal hash family.

We use our counters to keep track of an approximation of $\text{Count}(i,T)$. Initially, all the counters are set to 0. To process a pair (i_t, c_t), we compute $H_a(i_t)$ for each a with $1 \le a \le k$ and increment $C_{a,H_a(i_t)}$ by c_t. Let $C_{a,j}(T)$ be the value of the counter $C_{a,j}$ after processing X_1 through X_T. We claim that, for any item, the smallest counter associated with that item is an upper bound on its count, and with bounded probability the smallest counter associated with that item is off by no more than ε times the total count of all the pairs (i_t, c_t) processed up to that point. Specifically, we have the following theorem.

Theorem 13.12: *For any i in the universe U and for any sequence $(i_1, c_1), \ldots, (i_T, c_T)$,*

$$\min_{j=H_a(i),1\le a\le k} C_{a,j}(T) \ge \text{Count}(i,T).$$

Furthermore, with probability $1-(k/m\varepsilon)^k$ over the choice of hash functions,

$$\min_{j=H_a(i),1\le a\le k} C_{a,j}(T) \le \text{Count}(i,T) + \varepsilon \sum_{t=1}^{T} c_t.$$

Proof: The first bound,

$$\min_{j=H_a(i),1\le a\le k} C_{a,j}(T) \ge \text{Count}(i,T),$$

is trivial. Each counter $C_{a,j}$ with $j = H_a(i)$ is incremented by c_t when the pair (i, c_t) is seen in the stream. It follows that the value of each such counter is at least $\text{Count}(i,T)$ at any time T.

For the second bound, consider any specific i and T. We first consider the specific counter $C_{1,H_1(i)}$ and then use symmetry. We know that the value of this counter is at least $\text{Count}(i,T)$ after the first T pairs. Let the random variable Z_1 be the amount the counter is incremented owing to items other than i. Let X_t be a random variable that is 1 if $i_t \ne i$ and $H_1(i_t) = H_1(i)$: X_t is 0 otherwise. Then

$$Z_1 = \sum_{\substack{t:1\le t\le T, i_t\ne i \\ H_1(i_t)=H_1(i)}} c_t = \sum_{t=1}^{T} X_t c_t.$$

Because H_1 is chosen from a 2-universal family, for any $i_t \ne i$ we have

$$\Pr(H_1(i_t) = H_1(i)) \le \frac{k}{m}$$

and hence

$$\mathbf{E}[X_t] \le \frac{k}{m}.$$

It follows that

$$\mathbf{E}[Z_1] = \mathbf{E}\left[\sum_{t=1}^{T} X_t c_t\right] = \sum_{t=1}^{T} c_t \mathbf{E}[X_t] \le \frac{k}{m}\sum_{t=1}^{T} c_t.$$

By Markov's inequality,

$$\Pr\left(Z_1 \geq \varepsilon \sum_{t=1}^{T} c_t\right) \leq \frac{k/m}{\varepsilon} = \frac{k}{m\varepsilon}. \tag{13.2}$$

Let $Z_2, Z_3, \ldots, Z_k$ be corresponding random variables for each of the other hash functions. By symmetry, all of the Z_i satisfy the probabilistic bound of Eqn. (13.2). Moreover, the Z_i are independent, since the hash functions are chosen independently from the family of hash functions. Hence

$$\Pr\left(\min_{j=1}^{k} Z_j \geq \varepsilon \sum_{t=1}^{T} c_t\right) = \prod_{j=1}^{k} \Pr\left(Z_j \geq \varepsilon \sum_{t=1}^{T} c_t\right) \tag{13.3}$$

$$\leq \left(\frac{k}{m\varepsilon}\right)^k. \tag{13.4}$$

■

It is easy to check using calculus that $(k/m\varepsilon)^k$ is minimized when $k = m\varepsilon/\mathrm{e}$, in which case

$$\left(\frac{k}{m\varepsilon}\right)^k = \mathrm{e}^{-m\varepsilon/\mathrm{e}}.$$

Of course, k needs to be chosen so that k and m/k are integers, but this does not substantially affect the probability bounds.

We can use a count-min filter to track heavy hitters in the routing setting as follows. When a pair (i_T, c_T) arrives, we update the count-min filter. If the minimum hash value associated with i_T is at least the threshold q for heavy hitters, then we put the item into a list of potential heavy hitters. We do not concern ourselves with the details of performing operations on this list, but note that it can be organized to allow updates and searches in time logarithmic in its size by using standard balanced search-tree data structures; alternatively, it could be organized in a large array or a hash table.

Recall that we use Q to represent the total traffic at the end of the day.

Corollary 13.13: *Suppose that we use a count-min filter with $k = \lceil \ln \frac{1}{\delta} \rceil$ hash functions, $m = \lceil \ln \frac{1}{\delta} \rceil \cdot \lceil \frac{\mathrm{e}}{\varepsilon} \rceil$ counters, and a threshold q. Then all heavy hitters are put on the list, and any source–destination pair that corresponds to fewer than $q - \varepsilon Q$ bytes is put on the list with probability at most δ.*

Proof: Since counts increase over time, we can simply consider the situation at the end of the day. By Theorem 13.12, the count-min filter will ensure that all true heavy hitters are put on the list, since the smallest counter value for a true heavy hitter will be at least q. Further, by Theorem 13.12, the smallest counter value for any source–destination pair that corresponds to fewer than $q - \varepsilon Q$ bytes reaches q with probability at most

$$\left(\frac{k}{m\varepsilon}\right)^k \leq \mathrm{e}^{-\ln(1/\delta)} = \delta.$$

■

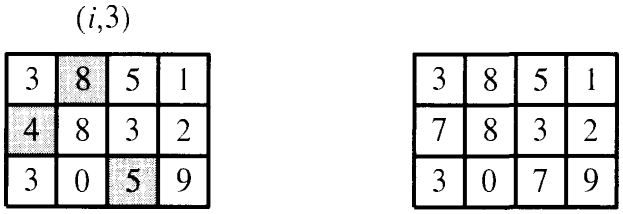

Figure 13.1: An item comes in, and 3 is to be added to the count. The initial state is on the left; the shaded counters need to be updated. Using conservative update, the minimum counter value 4 determines that all corresponding counters need to be pushed up to at least $4 + 3 = 7$. The resulting state after the update is shown on the right.

The count-min filter is very efficient in terms of using only limited randomness in its hash functions, only $O\left(\frac{1}{\varepsilon} \ln \frac{1}{\delta}\right)$ counters, and only $O\left(\ln \frac{1}{\delta}\right)$ computations to process each item. (Additional computation and space might be required to handle the list of potential heavy hitters, depending on its representation.)

Before ending our discussion of the count-min filter, we describe a simple improvement known as *conservative update* that often works well in practice, although it is difficult to analyze. When a pair (i_t, c_t) arrives, our original count-min filter adds c_t to each counter $C_{a,j}$ that the item i hashes to, thereby guaranteeing that

$$\min_{j=H_a(i), 1\le a\le k} C_{a,j}(T) \ge \text{Count}(i, T)$$

holds for all i and T. In fact, this can often be guaranteed without adding c_t to each counter. Consider the state after the $(t-1)$th pair has been processed. Suppose that, inductively, up to that point we have, for all i,

$$\min_{j=H_a(i), 1\le a\le k} C_{a,j}(t-1) \ge \text{Count}(i, t-1).$$

Then, when (i_t, c_t) arrives, we need to ensure that

$$C_{a,j}(t) \ge \text{Count}(i_t, t)$$

for all counters, where $j = H_a(i_t)$, $a \le l \le k$. But

$$\text{Count}(i_t, t) = \text{Count}(i_t, t-1) + c_t \le \min_{j=H_a(i_t), 1\le a\le k} C_{a,j}(t-1) + c_t.$$

Hence we can look at the minimum counter value v obtained from the k counters that i_t hashes to, add c_t to that value, and increase to $v + c_t$ any counter that is smaller than $v + c_t$. An example is given in Figure 13.1. An item arrives with a count of 3; at the time of arrival, the smallest counter associated with the item has value 4. It follows that the count for this item is at most 7, so we can update all associated counters to ensure they are all at least 7. In general, if all the counters i_t hashes to are equal, conservative update is equivalent to just adding c_t to each counter. When the i_t are not all equal, the conservative update improvement adds less to some of the counters, which will tend to reduce the errors that the filter produces.