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

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

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 $1 \leq a \leq k$ and $0 \leq j \leq m/k - 1$, 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 $1 \leq a \leq k$, where $H_a : U \rightarrow [0, m/k - 1]$. 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 $(i_1, c_1), \dots, (i_T, c_T)$,*

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

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,

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

Let $Z_2, Z_3, \dots, Z_k$ 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

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

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

■

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

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

■

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 $(i_1, c_1), \dots, (i_T, c_T)$,*

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

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,

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

Let $Z_2, Z_3, \dots, Z_k$ 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

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

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

■

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

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

■