**CS 761: Randomized Algorithms** Fall 2019

# Lecture 4 — September 27, 2019

*Prof. Gautam Kamath* *By: Jesse Elliott, Lingyi Zhang*
*Edited by Vedat Levi Alev*

**Disclaimer:** These notes have not been subject to the usual scrutiny reserved for formal publications.

## 1 Review of Previous Lecture

### 1.1 Hash Functions

Recall that a hash function is a mapping $h : U \to V$, where $|U| = m$ is very large and $|V| = n < m$. Last time we talked about idealized hash functions which are assumed to be truly random.

**Definition 1.** *([1]) A hash function $h : U \to V$ is* ***truly random*** *if for all $x \in U$, independent of all $y \neq x \in U$,*

$$\mathbb{P}[h(x) = t] = \frac{1}{m}.$$

Truly random hash functions are convenient mathematically but, unfortunately, unrealistic. As discussed in the last lecture, picking a random function is not hard but representing a random function is hard. To see why this is the case, consider the following question.

**Question 1.** *How many functions from a set of $m$ objects to a set of $n$ objects exist?*

Each of the $m$ domain objects have $n$ choices, so to speak. Each domain object can be mapped to any of the $n$ range objects. Considering every object in the domain, we have $n^m$ possible functions. Hence, the number of bits required to represent such a function is $\lg n^m = m \lg n$ bits, which is large because we assumed that $m$ is large.

Today we will discuss functions that can be described more concisely, where the total number of functions is polynomial in $m$. This way we only need $\lg m^c = c \lg m = O(\lg m)$ bits, and therefore we can store and compute these functions much more easily.

## 2 $k$-Wise Independence

**Definition 2.** *We say that random variables $x_1, \ldots, x_n$ are $k$-wise independent when for all $I \subset [n]$, with $|I| \leq k$, for any $x_i^{'}s$,*

$$\mathbb{P}\left[\bigcap_{i \in I}(X_i = x_i)\right] = \prod_{i \in I} \mathbb{P}[X_i = x_i].$$

*When $k = 2$ we call this pairwise independence. When $k = n$ this is just full independence.*

**Example 1.** *Let* $X_1, X_2 \sim$*Ber*$(\frac{1}{2})$ *be random variables and let* $X_3 = X_1 + X_2 (\text{mod } 2)$ *be another random variable.*

**Claim 3.** $X_1, X_2,$ *and* $X_3$ *are pairwise independent.*

*Proof.* Indeed, notice that

$$\mathbb{P}[X_3 = 1 | X_1 = 0] = \mathbb{P}[X_3 = 1 | X_1 = 1] = \frac{1}{2} \Rightarrow \mathbb{P}[X_3 = 1] = \frac{1}{2}.$$

Therefore,

$$\begin{aligned} \mathbb{P}[X_3 = x_3 \wedge X_1 = x_1] &= \mathbb{P}[X_3 = x_3 | X_1 = x_1] \cdot \mathbb{P}[X_1 = x_1] \\ &= \frac{1}{2} \cdot \frac{1}{2} \\ &= \mathbb{P}[X_3 = x_3] \cdot \mathbb{P}[X_1 = x_1]. \end{aligned}$$

□

**Example 2.** *(Generalization of Example 1) Let* $X_1, \ldots, X_b \sim$*Ber*$(\frac{1}{2})$ *be independent random variables and let* $Y_S = \sum_{i \in S} X_i \ (\text{mod } 2)$, *with* $\emptyset \neq S \subset [b]$ *(notice there are* $2^b - 1$ $Y_S^{'}s$*).*

**Claim 4.** *(Not proven) Let* $S_1$ *and* $S_2$ *be non-empty subsets of* $[b]$*. Then* $Y_{S_1}$ *and* $Y_{S_2}$ *are pairwise independent.*

Since there are $2^b - 1$ possible subsets $S$, the collection $\{Y_S\}_{\emptyset \neq S \subset [b]}$ contains $2^b - 1$ random variables. We started with $b$ random bits $\{X_i\}_{i=1}^b$ and ended up with a collection $\{Y_S\}_{\emptyset \neq S \subset [b]}$ containing $2^b - 1$ random variables. This is an exponential increase in randomness. Although the randomness is only pairwise independence, as it turns out, pairwise independence is very powerful. As we will see in the next example, Chebyshev still holds under pairwise independence. And, as we will see later in the course, pairwise independence is very useful in derandomization.

**Example 3.** *(Chebyshev's Inequality Holds Under Pairwise Independence) Let* $X_1, \ldots, X_n$ *be pairwise independent random variables and let* $X = \sum_{i=1}^n X_i$*. Then,*

$$\begin{aligned} Var[X] &= \sum_{i=1}^n Var[X_i] + \sum_{i<j} 2Cov[X_i, X_j] \\ &= \sum_{i=1}^n Var[X_i] + \sum_{i<j} 2[\mathbb{E}[X_i X_j] - \mathbb{E}[X_i]\mathbb{E}[X_j]] \\ &= \sum_{i=1}^n Var[X_i] + \sum_{i<j} 2[\mathbb{E}[X_i]\mathbb{E}[X_j] - \mathbb{E}[X_i]\mathbb{E}[X_j]], \qquad \textit{by pairwise independence,} \\ &= \sum_{i=1}^n Var[X_i] + 0 \\ &= \sum_{i=1}^n Var[X_i], \end{aligned}$$

*and therefore, when assuming pairwise independence, the variance of the sum is equal to the sum of the variances. Now, recall that Chebyshev tells us that*

$$\begin{aligned}\mathbb{P}[|X - \mathbb{E}[X]| \geq a] &\leq \frac{Var[X]}{a^2}, \qquad \text{and since we are assuming pairwise independence,} \\ &= \sum_{i=1}^{n} \frac{Var[X_i]}{a^2}.\end{aligned}$$

*This is all we mean when we say that Chebyshev holds for pairwise independent random variables. It holds in the sense that, even if there is some complicated correlation structure between the $X_i^{'}s$, assuming pairwise independence, Chebyshev style arguments can be applied as we did before.*

**Remark 5.** *Chernoff style arguments will not necessarily work under pairwise independence.*

## 3 Universal Hash Function Families

### 3.1 Definitions

**Definition 6.** *Let $x_1, \ldots, x_k \in U, y_1, \ldots, y_k \in V$, with $x_1, \ldots, x_k$ distinct, $|U| = m$ and $|V| = n$. Then, $\mathscr{H}$ is a **strongly $k$-universal family of hash functions** if whenever $h$ is chosen uniformly at random from $\mathscr{H}$,*

$$\mathbb{P}[(h(x_1) = y_1) \wedge (h(x_2) = y_2) \wedge \ldots \wedge (h(x_k) = y_k)] = \frac{1}{n^k}.$$

Note the similarity to the standard hashing property (i.e. $\mathbb{P}[h(x) = j] = \frac{1}{n}, \ \forall j \in \{0, \ldots, n-1\}$). Now the independence property is restricted to subsets to size $k$. We will again mostly focus on the case when $k = 2$ because this already gives us a lot of power.

We will use a weaker property than strongly universal. We want to analyze the probability of collisions. The next definition, which is weaker than strongly $k$-universal, is only going to attempt to reason about the probability of collisions.

**Definition 7.** *Let $x_1, \ldots, x_k \in U$ be distinct. Then, $\mathscr{H}$ is a **$k$-universal family of hash functions** if whenever $h$ is chosen uniformly at random from $\mathscr{H}$,*

$$\mathbb{P}[h(x_1) = h(x_2) = \ldots = h(x_k)] \leq \frac{1}{n^{k-1}}.$$

To see why the probability is bounded by $\frac{1}{n^{k-1}}$, consider the case when $k = 2$. Then,

$$\mathbb{P}[h(x_1) = h(x_2)] \leq \frac{1}{n}.$$

**Remark 8.** *Strongly $k$-universal implies $k$-universal.*

We will again mostly focus on the case when $k = 2$.

Next, we will discuss why these definitions are useful, and then we will look at examples to construct them.

## 3.2 Why Are These Definitions Useful?

We will see that these definitions give us the properties that we wanted from truly random hash functions. In particular, we wanted

1. Small expected look up time.
2. Small worst case look up time.

We will prove the exact same guarantee for 1 but only a weaker guarantee for 2.

### 3.2.1 Expected Look Up Time

Recall the set up for hashing from Lecture 3, where we have a mapping $h : U \to V$, a set $S = \{x_1, \ldots, x_{|S|}\} \subset U$ with $|U| = m$ and $|V| = n$. And, for some element $x$, we want to check if $x \in S$. That is, we want to implement a set data structure using hash functions. Let $X_i$ be an indicator random variable such that

$$X_i = \left\{ \begin{array}{ll} 1 & \text{if } h(x) = h(x_i), \\ 0 & \text{otherwise.} \end{array} \right\}$$

Suppose we want to look up $x \in U$. There are two cases to consider.

1. If $x \notin S$ then $\mathbb{P}[X_i = 1] \leq \frac{1}{n} \Rightarrow \mathbb{E}[X_1 + \ldots + X_{|S|}] \leq \frac{|S|}{n}$.
2. If $x \in S$ then we know there will be a collision and $\mathbb{E}[X_1 + \ldots + X_{|S|}] \leq 1 + \frac{|S|-1}{n}$.

Hence, if $|S| = O(n)$ then we have $O(1)$ look up time in expectation, which is the same guarantee we had for truly random hash functions.

### 3.2.2 Worst Case Look Up Time

Let $X_{ij}$ be an indicator random variable that indicates a collision between $X_i$ and $X_j$. That is,

$$X_{ij} = \left\{ \begin{array}{ll} 1 & \text{if } h(x_i) = h(x_j), \\ 0 & \text{otherwise.} \end{array} \right\}$$

Then, by the definition of a 2-universal family of hash functions, $\mathbb{E}[X_{ij}] \leq \frac{1}{n}$ and if $X = \sum_{i<j} X_{ij}$ then

$$\mathbb{E}[X] = \sum_{i<j} \mathbb{E}[X_{ij}] \leq \binom{|S|}{2} \cdot \frac{1}{n} = \frac{|S|^2}{2n}.$$

Now, since we only have pairwise independence, we will just apply Markov's inequality:

$$\mathbb{P}\left[X \geq \frac{|S|^2}{n}\right] \leq \frac{\frac{|S|^2}{2n}}{\frac{|S|^2}{n}} = \frac{1}{2}.$$

Now, we are going to look at the most loaded bin and consider the number of collisions in the most loaded bin. In particular, the number of collisions in the most loaded bin will be less then the number of overall collisions and this will give us something useful. Let $Y$ be the load of the most loaded bin (i.e. the bin where most elements are hashed to).

**Question 2.** *What is the number of collisions that are realized in a bin with $Y$ elements?*

The answer is $\binom{Y}{2}$, because every pair of two elements in $Y$ are a collision. Now,

$$\mathbb{P}\left[\binom{Y}{2} \geq \frac{|S|^2}{n}\right] \leq \mathbb{P}\left[X \geq \frac{|S|^2}{n}\right] \leq \frac{1}{2},$$

so that after simplification we have

$$\mathbb{P}\left[Y \geq 1 + |S|\sqrt{(2/n)}\right] \leq \frac{1}{2}.$$

Therefore if $|S| = n$ then $Y \leq 1 + \sqrt{2n}$, with probability $\geq \frac{1}{2}$.

We have managed to show that if we sample a hash function from a 2-universal family then the most loaded bin will have load bounded by $1 + \sqrt{2n}$ with probability $\geq 1/2$. Compare this to what we get with truly random hash functions: $O(\frac{\log n}{\log \log n})$. Hence, what we get is exponentially worse. However, it is still a non-trivial guarantee. And note that tighter bounds can be proven for larger values of $k$ (i.e. $k > 2$).

## 3.3 Examples

### 3.3.1 Strongly 2-universal hash functions from construction

We can construct a strongly 2-universal hash function far more easily than a truly random hash function. Rather than requiring, say, $m$ bits to specify a hash function, now we only require $\log m$ bits which is exponentially more succinct.

**Example 4.** *Let $p$ be a prime and let $|U| = |V| = p - 1$ and assume in this first example that $U = V = \{0, 1, \ldots, p-1\}$. Define*

$$h_{a,b}(x) := ax + b \pmod{p}, \text{ and } \mathscr{H} := \{h_{a,b} : 0 \leq a, b \leq p-1\},$$

*where $a$ and $b$ are integers sampled uniformly and randomly.*

**Claim 9.** *$\mathscr{H}$ is strongly 2-universal.*

*Proof.* We want to show that

$$\mathbb{P}[(h_{a,b}(x_1) = y_1) \text{ and } (h_{a,b}(x_2) = y_2)] = \frac{1}{p^2},$$

which holds if and only if

$$ax_1 + b = y_1 \pmod{p} \text{ and } ax_2 + b = y_2 \pmod{p},$$

which holds if and only if

$$a(x_2 - x_1) = y_2 - y_1 \pmod{p}.$$

Now to solve for $a$ we need that $(x_2 - x_1)^{-1}$ modulo $p$ exists, and hence we need that $\gcd(x_2 - x_1, p) = 1$ which is the case when $|x_2 - x_1| \leq p - 1$. Hence,

$$a = (y_2 - y_1)(x_2 - x_1)^{-1} \pmod{p}, \text{ with probability } \frac{1}{p}$$

and

$$b = (y_1 - ax_1) \pmod{p}, \text{ with probability } \frac{1}{p},$$

and thus the probability of picking both $a$ and $b$ is

$$\frac{1}{p} \cdot \frac{1}{p} = \frac{1}{p^2}.$$

□

Now, we want to generalize the construction to cases where the domain and the range are not the same size.

**Example 5.** *Let $p$ be a prime and let $d$ be a positive integer. This time let $|U| = p^d - 1$ and $|V| = p - 1$. Let $U = \{0, 1, \ldots, p^d - 1\}$ and $V = \{0, 1, \ldots, p - 1\}$. Interpret elements $\overline{u} \in U$ as vectors $\overline{u} = (u_0, \ldots, u_{d-1}), 0 \leq u_i \leq p - 1$, where $\overline{u} = \sum_{i=0}^{d-1} u_i p^i$ (i.e. interpret $\overline{u}$ as a $p-$ary number. For any vector $\overline{a} = (a_0, \ldots, a_{k-1})$ with $0 \leq a_i \leq p - 1$ and $0 \leq i \leq d - 1$ and for $b$ with $0 \leq b \leq p - 1$, define*

$$h_{\overline{a},b}(x) := \left( \sum_{i=0}^{d-1} a_i x_i + b \right) \pmod{p}$$

*and*

$$\mathscr{H} := \{h_{\overline{a},b} : 0 \leq a_1, \ldots, a_{d-1}, b \leq p - 1\}.$$

**Claim 10.** *$\mathscr{H}$ is strongly 2-universal.*

*Proof.* We need to prove that

$$\mathbb{P}[h_{\overline{a},b}(x_1) = y_1 \wedge h_{\overline{a},b}(x_2) = y_2] = \frac{1}{p^2}.$$

Assume $x_{1,0} \neq x_{2,0}$. Then, these conditions are equivalent to

$$a_0 x_{1,0} + b = (y_1 - \sum_{j=0}^{d-1} a_j x_{1,j}) \pmod{p}, \text{ and}$$
$$a_0 x_{2,0} + b = (y_2 - \sum_{j=0}^{d-1} a_j x_{2,j}) \pmod{p}.$$

This is again two equations having a unique solution. Hence, for every $a_0, \ldots, a_{d-1}$, there is exactly one choice of $(a_0, b)$ out of $p^2$ possibilities satisfying the conditions. Therefore,

$$\mathbb{P}[h_{\overline{a},b}(x_1) = y_1 \wedge h_{\overline{a},b}(x_2) = y_2] = \frac{1}{p^2}.$$

□

We have come up with a strongly 2-universal hash function that can map from a larger domain to a smaller domain. Now let's consider how complicated it is to actually store it. We have $d + 1$ parameters and $p$ choices for each and therefore we have $p^{d+1} \approx |U|$ possible hash functions in $\mathscr{H}$. The number of possible hash functions is comparable to the size of the universe. Therefore to represent a single hash function we require $O(\log |U|)$ bits.

## 3.4 Perfect Hashing

Even if the set that we want to hash is known beforehand, it is not clear how to easily and efficiently design a function that actually maps it to a range of comparable size. Now we will discuss how to accomplish this using 2-universal hash functions. The set up will be in terms of the **static dictionary problem**. A dictionary is an abstract data type that maintains a set $S$ of items (we assume that $|S| = m$). We only need the dictionary to support a look up operation. We want:

1. $O(1)$ look up time in the worst case, and
2. $n = O(m)$ space.

The difference now is that we will build our hash function based on having $S$ beforehand whereas in the hashing setting the function was specified first.

## 3.5 Easy Solution

To convey the idea, let's first assume that $n = m^2$. Pick a hash function uniformly at random from a 2-universal family. Let $X$ be a random variable equal to the number of collisions. We have proven that

$$\mathbb{P}\left[X \geq \frac{m^2}{n}\right] \leq \frac{1}{2}.$$

Therefore if $n = 2m^2$, there are no collisions with probability at least $\frac{1}{2}$. Hence, we can just repeat until we get a hash function with no collisions. This solution gives us $O(1)$ look up time, because there are no collisions, and $n = O(m^2)$ space. But this is too much space.

## 3.6 Two Level Solution

Now the plan is to assume $m = \Theta(n)$ and choose a hash function with only a few collisions. And, with any bin that has a collision, we will then run the easy solution as a second level of hashing. That is, we will hash again on the second level in such a way that there are no collisions. We will see that this solution requires only $O(m)$ space.

1. Choose a hash function $h_1$ from a 2-universal family with $m = \Theta(n)$. Repeat choosing $h_1$ until the number of collisions is less than or equal to $m$. Then,

$$\mathbb{P}\left[X \geq \frac{m^2}{n}\right] \leq \frac{1}{2} \Rightarrow \mathbb{P}[X \geq m] \leq \frac{1}{2}.$$

2. Now, for each bucket in the hash table, for each $i$ between 0 and $m-1$, let $c_i$ be the number of items in that bucket. Choose a hash function $h_{2,i} : U \to \{0, 1, \ldots, c_i^2\}$ uniformly and randomly from a 2-universal family. Randomly choose $h_{2,i}$ until there are no collisions.

Now the space used is $m$ for the first table and $\sum_{i=1}^{m} c_i^2$ for the second. That is,

$$\begin{aligned} m + \sum_{i=1}^{m} c_i^2 &\le m + 2\sum \binom{c_i}{2} + \sum_{i=1}^{m} c_i, && \text{where } \sum \binom{c_i}{2} \text{ is the number of collisions,} \\ &\le m + 2m + m, && \text{because the number of collisions is bounded by } m, \\ &= 4m. \end{aligned}$$

We have shown that if we have a known set $S$, then we can make a data structure with $O(1)$ look up time in the worst case and which takes the same amount of space up to a constant factor overhead. This is perfect hashing.

## 4 Streaming

In this section we will study some interesting problems including heavy hitters and distinct elements. We will see that ideas from hashing and $k$-wise independence are very helpful in this setting.

In some application where we have a massive data set, and the data is too much that we are not afford to store them all or read them once. In streaming setting, the algorithm can take one pass on the data but has very limited space for computation.

Since we have so little space, we cannot hope to solve the problem exactly, and we work with a weaker requirement that allows some error probability.

### 4.1 Heavy Hitters

A data stream is an array of element $X_1, ..., X_T$ arrive sequentially. For $t$-th element $X_t = (i_t, c_t)$, where $i_t$ is the identifier and $c_t$ is the weight.

Our goal is to find all the heavy hitters of the data stream.

Let $count(i) = \sum\limits_{i_t = i} c_t$ and the total weight $Q = \sum\limits_{t=1}^{T} c_t$.

**Definition 11.** *Item $i$ is $q$-heavy hitter if $count(i) \ge q$.*

For the heavy hitters problem, we want to report all heavy hitters in the stream and report elements that are not heavy hitters with a small error. Since we have limited space for computation, the space complexity we want to achieve here is $O(\log T)$ space or less. The algorithm we will introduce here to solve heavy hitter problem is Count Min-Sketch.

#### 4.1.1 Main idea

The idea of the Count Min-Sketch is to use a collection of independent hash functions to hash each element in the stream, keeping track of the number of times each bucket is hashed to. The hope is that for each heavy hitter, one of the buckets it hashes to witnesses few hashes from other elements. We would like to ensure the followings:

1. All $q$-Heavy hitters are reported.
2. If $count(i) \leq q - \varepsilon Q$, report with probability $\leq \delta$.
   (we want $q$ to be big, i.e. $q > \frac{1}{100}Q$. Otherwise, the problem is trivial)

### 4.1.2 Algorithm

To approach this problem, we will use a Bloom filter data structure with counters. We maintain a total number of $k$ hash functions and $kl$ counters.

We hash the element by its id and increment the counter by its weight. Since we are using a bloom filter, all id $i$ will be mapped to the same entry in the hash table. However, elements with different ids will also be mapped to the same entry in the same hash table due to collision, so we report the minimum to reduce the error.

1. Create $k$ hash tables, where each hash table has size $l$.
2. Initialize all entries to be 0.
3. $\forall t \in \{1, ..., T\}$; $\forall j \in [k]$, table $j$'s entry $h_j(i_t) = h_j(i_t) + c_t$
4. After reading all data, report all $i$ s.t. $\min\limits_{\substack{a=h_j(i)\\1\leq j\leq k}} \{$ counter in table $j$'s entry $a_j\} \geq q$ .i.e We report an id if the minimum counter associated with it is at least $q$.

**Example 6.**

| | | | | | |
|---|---|---|---|---|---|
| | | | $+c_t$ | | |
| $+c_t$ | | | | $+c_t$ | $+c_t$ |
| | $+c_t$ | | | | |
| | | $+c_t$ | | | |
| | | | | | |

### 4.1.3 Analysis

The correctness analysis can be split in two parts. In the first part, we will show that all the heavy hitters will be reported by the algorithm. In the second part, which is more complicated, we will show that the elements that are not heavy hitters will be reported with a small error probability.

1. All $q$-HH are reported.

   Noticed that since we are using a bloom filter to store counter, the id $i$ will always mapped to the same counter, and the counter will be at least as large as the sum of all the weights for this id. Thus, for a $q$-heavy hitter, all the counters associated to it must be at least $q$ and all $q$-HH are reported.

2. Suppose that an item $i$ has $count(i) \leq q - \varepsilon Q$. We want to show that it will be reported with probability $\leq \delta$

We first define the following notation for each table entry:

$$c_{j,a} = \text{count in table } j\text{'s entry a}$$

Then we have:

$$c_{j,h_j(i)} = \text{count}(i) + Z_j \text{ where } Z_j \text{ are contribution to } c_j, h_j(i) \text{ from "not i"}$$

By the definition of 2-universal hash function, we can get

$$\mathbb{P}[\text{Item } x \text{ and } y \text{ collide in h.t } j] \leq \frac{1}{l};$$

Then the expected value for $Z_j$ is

$$\mathbb{E}[Z_j] \leq \frac{Q}{l};$$

By Markov inequality, we can obtain:

$$\mathbb{P}[Z_j \geq \varepsilon Q] \leq \frac{\frac{Q}{l}}{\varepsilon Q} = \frac{1}{\varepsilon l};$$

Since we report the minimum of the $k$ hash tables, we have

$$\mathbb{P}[\forall j, Z_j \geq \varepsilon Q] \leq \left(\frac{1}{\varepsilon l}\right)^k;$$

The goal is to get:

$$\mathbb{P}[\forall j, Z_j \geq \varepsilon Q] \leq \left(\frac{1}{\varepsilon l}\right)^k \leq \delta;$$

This can be achieved when we choose $l = \frac{l}{\varepsilon}, k = \log\left(\frac{1}{\delta}\right)$.

For the Count Min-Sketch algorithm, we just need to maintain $O(kl)$ counters for this task. The hash functions can be stored in $O(k)$ words. Note that the total required space for the algorithm is $O(kl)$ words, i.e. $O\left(\frac{log(1/\delta)}{\varepsilon}\right)$.

## 4.2 Distinct Elements

The input is a sequence $(a_1, .., a_n)$ of indices with each $a_i \in \{1, ..., m\}$. Define $D = \{a_i : i = 1, ..., m\}$ be the distinct elements that occur in the stream. The goal is to count the number of distinct elements in the stream in one pass with very limited space, and return a good estimate on the number of the distinct elements in the steam.

Note that it takes $O(\log m)$ memory to represent each distinct element.

**Observation 12.** *If $m > n$, simply use a hash function to map input stream to $n^2$. There's no collisions with high probability, so $m$ is irrelevant.*

In order to make this problem actually interesting, we will assume $n \gg m$.

### 4.2.1 Basic Idea

Before giving out the algorithm, we first think of mapping all elements in the universe to an interval $[0, 1]$. The idea is to use a hash function that map each element to points evenly distributed on the interval. For $D$ distinct elements, we want the smallest hashed value maps close to $\approx \frac{1}{D}$ and the $t$-smallest hashed value maps close to $\approx \frac{t}{D}$. Then we can estimate the value of $D$ with the value $t$ and the $t$-th smallest hashed value by $D \approx \frac{t}{t\text{-th smallest hashed value}}$.

### 4.2.2 Real Algorithm

Instead of mapping the hashed value to interval $[0, 1]$, we change the interval to $\{1, ..., l\}$. We want to pick a hash function that creates no collision in high probability, and ideally, we want the hash values to be evenly distributed in values in $\{1, ..., l\}$.

Since we have limited space, we do not want to store all the hashed values. We only want to store $t$ numbers, corresponds to the $t$ smallest values. Same as the basic idea stated above, we just need to know the hash value $T$ for the $t$-th smallest hashed value, we can estimate $D$.

1. Pick a hash function $h$ that is strongly 2-universal.
2. Compute $h(a_i)$ for each $a_i$ in stream, and maintain a list of $t$ smallest.
3. Return $Y = \frac{tl}{T}$, where $T$ is $t$-th smallest hashed value.

Note that $Y$ is an estimate of $D$.

### 4.2.3 Anaylsis

In order to show we can obtain a good estimate of $D$ by the above algorithm, we first show that we have $(1 - \varepsilon)D \le Y \le (1 + \varepsilon)D$ with constant probability.

**Theorem 13.** $\mathbb{P}[(1 - \varepsilon)D \le Y \le (1 + \varepsilon)D] \ge \frac{2}{3}$.

*Proof.* Before trying to prove $\mathbb{P}[(1 - \varepsilon)D \le Y \le (1 + \varepsilon)D] \ge \frac{2}{3}$, we will break it into two cases $\mathbb{P}[Y \ge (1+\varepsilon)D] \le \frac{1}{6}$ and $\mathbb{P}[Y \le (1-\varepsilon)D] \le \frac{1}{6}$. Note that if we are able to show $\mathbb{P}[Y \ge (1+\varepsilon)D] \le \frac{1}{6}$ and $\mathbb{P}[Y \le (1 - \varepsilon)D] \le \frac{1}{6}$, it is sufficient to show $\mathbb{P}[(1 - \varepsilon)D \le Y \le (1 + \varepsilon)D] \ge \frac{2}{3}$.

**Case 1:** $\mathbb{P}[Y \ge (1 + \varepsilon)D] \le \frac{1}{6}$.

Supposing the output $y \sim Y$ is at least $(1 + \varepsilon)D$, i.e. $Y \ge (1 + \varepsilon)D$.

**Remark 14.** *From the algorithm, we know that* $Y = \frac{tl}{T}$.

When $Y \ge (1 + \varepsilon)D$, we can get

$$\begin{aligned} T &\le \frac{tl}{(1+\varepsilon)D} \\ &\le \frac{tl\left(1 - \frac{\varepsilon}{2}\right)}{D} \text{ (holds for small } \varepsilon\text{).} \end{aligned}$$

Thus, we also have $\mathbb{P}[Y \geq (1+\varepsilon)D] \leq \mathbb{P}\left[T \leq \frac{tl\left(1-\frac{\varepsilon}{2}\right)}{D}\right]$.

Note that, if we can show $\mathbb{P}\left[T \leq \frac{tl\left(1-\frac{\varepsilon}{2}\right)}{D}\right] \leq \frac{1}{6}$, we can also show $\mathbb{P}[Y \geq (1+\varepsilon)D] \leq \frac{1}{6}$.

In order to let $T \leq \frac{tl(1-\frac{\varepsilon}{2})}{D}$, we need at least $t$ items hashed to below $\frac{tl\left(1-\frac{\varepsilon}{2}\right)}{D}$.

Let's define an indicator

$$X_i = \begin{cases} 1 & \text{if } h(a_i) \leq \frac{tl\left(1-\frac{\varepsilon}{2}\right)}{D}; \\ 0 & \text{otherwise.} \end{cases}$$

Let $X = \sum_{i=1}^{D} X_i$, then we can get the expected value

$$\mathbb{E}[X] = \sum_{i=1}^{D} \mathbb{E}[X_i] = D \cdot \mathbb{E}[X_i] \leq D \cdot \frac{tl\left(1-\frac{\varepsilon}{2}\right)}{D}.$$

**Remark 15.** *For Bernoulli distribution, we have:* $Var[Ber(p)] = p(1-p) \leq p$.

In order to bound the derivation, we want to use Chebyshev's inequality. Thus, we calculate the variance

$$\begin{aligned} Var[X] &= \sum_i Var(X_i) \text{ (by pairwise independence)} \\ &\leq D \cdot \frac{tl\left(1-\frac{\varepsilon}{2}\right)}{D} \\ &= t\left(1-\frac{\varepsilon}{2}\right). \end{aligned}$$

By applying Chebyshev's inequality, we get:

$$\begin{aligned} \mathbb{P}[X \geq t] &= \mathbb{P}\left[X \geq t(1-\frac{\varepsilon}{2}) + \frac{\varepsilon}{2}t\right] \\ &\leq \mathbb{P}\left[|X - \mathbb{E}[X]| \geq \frac{\varepsilon}{2}t\right] \\ &\leq \frac{t\left(1-\frac{\varepsilon}{2}\right)}{\left(\frac{\varepsilon}{2}t\right)^2} \\ &= \frac{4}{\varepsilon^2 t} && \text{(set } t = O(1/\varepsilon^2)\text{)} \\ &\leq \frac{1}{6}. \end{aligned}$$

**Cases 2:** $\mathbb{P}[Y \leq (1-\varepsilon)D] \leq \frac{1}{6}$.

Same as case 1.

□

**Remark 16.** *From above, we have $(1-\varepsilon)D \leq Y \leq (1+\varepsilon)D$ with probability $\geq \frac{2}{3}$. It was shown in homework 1 problem 1 that, by using Chernoff bounds, we can achieve $\delta$ failure probably by taking the median of $O(\log 1/\delta)$ independent repetitions of the algorithm.*

### 4.2.4 Runtime

Note that we store $t = 1/\varepsilon^2$ numbers in the algorithm in total. Since it requires $\log l$ bits for each number, we need $O(\log m)$ space to store all numbers ($l = O(m^3)$).

Total space is $O\left(\frac{1}{\varepsilon^2}\log m \log(1/\delta)\right)$ and time per update is $O(\log(1/\varepsilon))$.

**Fact 17.** *Best algorithm known so far ([2]) uses $O\left(\frac{1}{\varepsilon^2} + \log m\right)$ space and $O(1)$ for update.*

## References

[1] Erik Demaine. Advanced data structures, lecture 10, 6.897. page 1. `https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-851-advanced-data-structures-spring-2012/calendar-and-notes/MIT6_851S12_L10.pdf`, 2012.

[2] Daniel M. Kane, Jelani Nelson, and David P. Woodruff. An optimal algorithm for the distinct elements problem. In *Proceedings of the Twenty-ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '10, pages 41–52, New York, NY, USA, 2010. ACM.