Algorithms for Big Data (III) (Fall 2020)

Instructor: Chihao Zhang
Scribed by: Guoliang Qiu

Last modified on Oct 20, 2020

Today, we first review the construction of the universal hash family and provide a construction of strongly universal hash family. Then we introduce two algorithms to count the number of distinct elements in the streaming model.

1 Review the Construction of the Universal Hash Function Families

1.1 Universal Hash Function Families

Let $\mathcal{H}$ be a family of functions from $[m]$ to $[n]$ where $m \geq n$. We call $\mathcal{H}$ k-universal if for every distinct $x_1, \cdots, x_k \in [m]$, we have

$$\mathbf{Pr}_{h\in\mathcal{H}}[h(x_1) = h(x_2) = \cdots = h(x_k)] \leq \frac{1}{n^{k-1}}$$

Moreover, we call $\mathcal{H}$ strongly k-universal if for every distinct $x_1, \cdots, x_k \in [m]$ and $y_1, \cdots, y_k \in [n]$, we have

$$\mathbf{Pr}_{h\in\mathcal{H}}\left[\bigwedge h(x_i) = y_i\right] = \frac{1}{n^k}.$$

1.2 The Construction of 2-Universal Hash Family

Suppose we want to construct the hash functions from $[m]$ to $[n]$. First, we choose a prime $p \geq m$ and let

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

Last time, we have already proved the following theorem.

Theorem 1. *$\mathcal{H} = \{h_{a,b} : 1 \leq a \leq p-1, 0 \leq b \leq p-1\}$ is a 2-universal hash family.*

In the next section, we first show how to turn the previous construction into a strongly 2-universal hash family.

2 Strongly 2-Universal Hash Family

2.1 Base Case

First, when $m = n = p$ are primes, then we can modify the previously constructed $\mathcal{H}$ to get a strongly 2-universal family. In this case, we have

Theorem 2. *$\mathcal{H} = \{h_{a,b}(x) = (ax + b) \mod p : 0 \le a, b \le p - 1\}$ is a strongly 2-universal family.*

To prove this theorem, we exploit the following lemma.

Lemma 3. *The equation $ax + b = 0 \mod p$ has unique solution (in $\mathbb{F}_p$) if $a \neq 0$ and p is a prime.*

Proof. The equations $h_{a,b}(x_1) = y_1$ and $h_{a,b}(x_2) = y_2$ are equivalent to

$$ax_1 + b = y_1 \mod p, \quad ax_2 + b = y_2 \mod p.$$

They have a unique solution

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

Therefore, we have

$$\mathbf{Pr}_{h_{a,b} \in \mathcal{H}} \left[h_{a,b}(x_1) = y_1 \wedge h_{a,b}(x_2) = y_2 \right] = \frac{1}{p^2}.$$

□

2.2 General Case

We can naturally generalize $m = p$ to $m = p^k$. Write every number x in base p, i.e.,

$$x = \sum_{i=0}^{k-1} x_i \cdot p^i.$$

Then for every $\bar{a} = (a_0, a_1, \cdots, a_{k-1})$, with $0 \le a_i \le p - 1$ and $0 \le b \le p - 1$, define

$$h_{\bar{a},b}(x) = \left(\sum_{i=0}^{k-1} a_i x_i + b \right) \mod p.$$

Theorem 4. *$\mathcal{H} := \{h_{\bar{a},b} : \bar{a} \in \mathbb{F}_p^k, b \in \mathbb{F}_p\}$ is a strongly 2-universal hash family.*

Proof. Let $x, y \in \mathbb{F}_{p^k}$ be two numbers differing on the position i in their base p form, that is, $x_i \neq y_i$. For every $u, v \in \{0, 1, \cdots, p - 1\}$, we have equations

$$\begin{cases} a_i x_i + b = (u - \sum_{j \neq i} a_j x_j) \mod p \\ a_i y_i + b = (v - \sum_{j \neq i} a_j y_j) \mod p \end{cases}$$

For fixed x, y, u, v and $\{a_j\}_{j \neq i}$, a unique pair (a_i, b) (out of p^2 pairs) is determined. Therefore,

$$\mathbf{Pr}_{h_{\bar{a},b} \in \mathcal{H}} \left[h_{\bar{a},b}(x) = u \wedge h_{\bar{a},b}(y) = v \right] = \frac{1}{p^2}.$$

□

3 Counting Distinct Elements

In this section, we discuss another interesting problem in the streaming model.

Recall that, we are given a sequence of numbers $\langle a_1, a_2, \cdots, a_m \rangle$ where each $a_i \in [n]$. The sequence of numbers defines a frequency vector $\boldsymbol{f} = (f_1, \cdots, f_n)$ where $f_i = |\{k \in [m] : a_k = i\}|$. We want to compute the number $d = |\{i \in [n] : f_i > 0\}|$. The value d is the number of distinct elements in the stream.

3.1 The AMS Algorithm

The algorithm is originally described by Flajolet and Martin and the version presented here is due to Alon, Matias and Szegedy.

We first introduce some notations. For every integer $p > 0$, we use zero(p) to denote number of trailing zeros of p in binary, i.e., $\text{zero}(p) \triangleq \max\left\{i \,:\, 2^i \text{divides } p\right\}$. The description of the algorithm is asfollows:

Algorithm 1 AMS Algorithms for Counting Distinct Elements

Init:
A random Hash function $h : [n] \to [n]$ from a 2-universal family.
$Z \leftarrow 0$
On Input y:
if $\text{zero}(h(y)) > Z$ **then**
  $Z \leftarrow \text{zero}(h(y))$
end if
Output:
$\hat{d} = 2^{Z+\frac{1}{2}}$.

In words, the algorithm calculates the maximum number r of trailing zeros of the inputs after applying certain Hash function and outputs $2^{r+\frac{1}{2}}$. Let us see why it works.

First observe that after applying the Hash function h, $h(y)$ is uniform in $[n]$. Therefore, what we did in the algorithm for each input y is equivalent to picking a number uniformly in $[n]$ and looking at its number of trailing zeros. If we assume $n = 2^k$ for some integer k, this is equivalent to generate a uniform length-k binary string and check the number of its trailing zeros. Therefore, the probability that it has s trailing zeros is 2^{-s}. Therefore, if we know the maximum number of trailing zeros is s among m uniform binary strings, 2^s should be a good guess of m.

3.2 The Analysis of the AMS Algorithm

In this section, we rigorously show that $Z \approx \log_2 d$.

For every $0 \le r \le \log_2 n$, we use a random variable Y_r to denote the number of $h(a_i)$ with trailing zeros at least r, or $Y_r \triangleq \{k \in [n] \,:\, f_k > 0 \wedge \text{zero}(h(k)) \ge r\}$. Notice that the sequence of variables $\{Y_r\}_{0 \le r \le n}$ determines the variable Z since $Z = \max_r\{Y_r > 0\}$. Therefore, it is enough to understand the behavior of Y_r.

For every $k \in [n]$, we denote $X_{k,r}$ as the indicator that $h(k)$ has at least r trailing zeros, then we can decompose that $Y_r = \sum_{k \in [n]: f_k > 0} X_{k,r}$. According to this decomposition, we have:

$$\mathbf{E}[Y_r] = \sum_{\substack{0 \le k \le n-1 \\ f_k > 0}} \mathbf{E}\left[X_{k,r}\right] = \frac{d}{2^r};$$

$$\mathbf{Var}[Y_r] = \sum_{\substack{0 \le k \le n-1 \\ f_k > 0}} \mathbf{Var}\left[X_{k,r}\right] = \frac{d}{2^r},$$

where the linearity of variance comes from the fact that $X_{k,r}$s are pairwise independent variables.

Applying Markov's inequality, we obtain

$$\mathbf{Pr}[Y_r > 0] = \mathbf{Pr}[Y_r \ge 1] \le \mathbf{E}[Y_r] = \frac{d}{2^r}.$$

Applying Chebyshev inequality, we obtain

$$\mathbf{Pr}[Y = 0] \le \mathbf{Pr}\left[|Y_r - \mathbf{E}[Y_r]| \ge \frac{d}{2^r}\right] \le \frac{2^r}{d}$$

We know that $Y_r > 0$ for all $r \le Z$ and $Y_r = 0$ for all $r > Z$. Therefore, Z cannot be too far from $\log_2 d$:

- if $Z \ll \log_2 d$, there exists r such that $r \gg \log_2 d$ with $Y_r = 0$, which happens with small probability;
- if $Z \gg \log_2 d$, there exists r such that $r \gg \log_2 d$ with $Y_r > 0$, which happens with small probability as well.

Let us bound the probabilities of above two events respectively. For the first event, if $\hat{d} \le \frac{d}{3}$, let $\hat{r} := \arg\max_{r \in \mathbb{N}} 2^{r+\frac{1}{2}} \le \frac{d}{3}$, then

$$\mathbf{Pr}\left[\hat{d} \le \frac{d}{3}\right] = \mathbf{Pr}[Z \le \hat{r}] = \mathbf{Pr}[Y_{\hat{r}+1} = 0] \le \frac{2^{\hat{r}+1}}{d} \le \frac{\sqrt{2}}{3}$$

For the second event, if $\hat{d} \ge 3d$, let $\hat{r} := \arg\min_{r \in \mathbb{N}} 2^{r+\frac{1}{2}} \ge 3d$.

$$\mathbf{Pr}\left[\hat{d} \ge 3d\right] = \mathbf{Pr}[Z \ge \hat{r}] = \mathbf{Pr}[Y_{\hat{r}} > 0] \le \frac{d}{2^{\hat{r}}} \le \frac{\sqrt{2}}{3}$$

Thus, we have $\mathbf{Pr}\left[\frac{d}{3} \le \hat{d} \le 3d\right] \ge 1 - \frac{2\sqrt{2}}{3}$.

This algorithm costs $O(\log n)$ bits of memory. Moreover, we can apply the standard Median trick to boost the success probability. Using $O(\log \frac{1}{\delta} \log n)$ bits of memory, we can obtain

$$\mathbf{Pr}\left[\frac{d}{3} \le \hat{d} \le 3d\right] \ge 1 - \delta.$$

3.3 The BJKST Algorithm

The performance of AMS has an obvious drawback: We can only guarantee that $\hat{d} \in [d/3, 3d]$. The reason for this coarse bound is simple: We already know that the expected number of uniformly distributed distinct elements before seeing the first one with r trailing zeros is 2^r. On the other hand, the expected number before seeing the first one with $r+1$ trailing zeros is 2^{r+1}. So when we see r trailing zeros, which number between 2^r and $2^{r+1} - 1$ should we guess? The AMS simply guesses $2^{r+\frac{1}{2}}$. In fact, if we can store more information on "how many distinct numbers seen so far has r trailing zeroes", the guess can be more accurate.

The following improvement of AMS is due to Bar-Yossef, Jayram, Kumar, Sivakumar, and Trevisan, which implements the idea above. See Algorithm 2.

The algorithm maintains a bucket B, which stores those y whose zeros($h(y)$) is larger than the current Z. In the BJKST algorithm we set $L = \frac{c}{\epsilon^2}$ for the size of B, the size of B captures the trade-off between memory consumption and the accuracy of the algorithm, two extreme cases are:

- if $L = \infty$, B stores all entries, and the algorithm is exact;
- if $L = 2$, the algorithm is equivalent to AMS.

Algorithm 2 BJKST Algorithms for Counting Distinct Elements

Init:
A random Hash functions $h : [n] \to [n]$ from a 2-universal families; $Z \leftarrow 0, B \leftarrow \emptyset$
On Input y:
if zero$(h(y)) > Z$ **then**
  $B \leftarrow B \cup \{(y, \text{zero}(h(y))\}$
  while $|B| \geq \frac{c}{\epsilon^2}$ **do**
    $Z \leftarrow Z + 1$
    Remove all (α, β) with $\beta < Z$ from β
  end while
end if
Output:
$\hat{d} = |B| \cdot 2^Z$.

3.4 The Analysis of the BJKST Algorithm

Recall $Y_r = \sum_{k \in [n]: f_k > 0} X_{k,r}$ is the number of $h(a_i)$ with at least r trailing zeros.

Suppose $Z = \hat{r}$ at the end of the algorithm, then $Y_{\hat{r}} = |B|$ and $\hat{d} = Y_{\hat{r}} \cdot 2^{\hat{r}}$. We use A to denote the bad event that $|Y_{\hat{r}} \cdot 2^{\hat{r}} - d| \geq \epsilon d$, or equivalently

$$\left| Y_{\hat{r}} - \frac{d}{2^{\hat{r}}} \right| = |Y_{\hat{r}} - \mathbf{E}[Y_{\hat{r}}]| \geq \frac{\epsilon d}{2^{\hat{r}}} = \epsilon \mathbf{E}[Y_{\hat{r}}]$$

We will bound the probability of A based on the following observation,

- if $\mathbf{E}[Y_{\hat{r}}]$ is large, then it is well concentrated.
- For smaller $\mathbf{E}[Y_{\hat{r}}]$, we show it is unlikely to happen.

$$
\begin{aligned}
\mathbf{Pr}[A] &= \sum_{t=1}^{\log_2 n} \mathbf{Pr}\left[\left|Y_{\hat{r}} - \frac{d}{2^{\hat{r}}}\right| \geq \frac{\epsilon d}{2^{\hat{r}}} \wedge \hat{r} = t\right] \\
&\leq \sum_{t=1}^{s-1} \mathbf{Pr}\left[\left|Y_t - \frac{d}{2^t}\right| \geq \frac{\epsilon d}{2^t}\right] + \sum_{t=s}^{\log_2 n} \mathbf{Pr}[\hat{r} = t] \\
&= \sum_{t=1}^{s-1} \mathbf{Pr}\left[\left|Y_t - \frac{d}{2^t}\right| \geq \frac{\epsilon d}{2^t}\right] + \mathbf{Pr}\left[Y_{s-1} > \frac{c}{\epsilon^2}\right] \\
&\leq \sum_{t=1}^{s-1} \frac{2^t}{\epsilon^2 d} + \frac{\epsilon^2 d}{c 2^{s-1}} \\
&\leq \frac{2^s}{\epsilon^2 d} + \frac{\epsilon^2 d}{c 2^{s-1}} \\
&= O(1) \text{ when choosing } 2^s = O(\epsilon^2 d).
\end{aligned}
$$

It remains to look at the memory consumption. We need to store the function h with $O(\log n)$ bits of memory the bucket B with $O\left(\frac{c}{\epsilon^2} \log n\right)$ bits of memory. After using the standard Median trick, we totally consume $O(\frac{c}{\epsilon^2} \cdot \log n \cdot \log \frac{1}{\delta})$ bits of memory.