CSE 521: Design and Analysis of Algorithms I **Winter 2017**

Lecture 4: Hashing and Streaming Algorithms

Lecturer: Shayan Oveis Gharan *01/18/2017* *Scribe: Yuqing Ai*

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

At the end of the previous lecture, we introduced the basic notions of hashing and saw some of its applications. In this lecture, we are going to study hashing in more detail.

4.1 The Problem of Hashing

Let $\mathcal{U} = \{1, 2, ..., |\mathcal{U}|\}$ be a large universe of numbers, let $X_1, ..., X_n \in \{1, 2, ..., |\mathcal{U}|\}$ be n input numbers in such a universe where $n \ll |\mathcal{U}|$. Recall from the last lecture, our goal is to construct a family of hash functions $\mathcal{H}$, where every function in this family maps from $\mathcal{U}$ to $\{1, 2, ..., N\}$. We want the probability of collision $\mathbb{P}_{h\sim\mathcal{H}}[\exists i, j \in \{1, ..., n\}, i \neq j, h(X_i) = h(X_j)]$ to be small. In the mean time, we would like to use as little space as possible.

Recall that in the example of birthday paradox, where we have n people and N days in a year, the probability that two people were born in the same day is small when $N \gg n^2$. Therefore, if we map the n input numbers uniformly to $\{1, 2, ..., cn^2\}$ for some large enough constant c, then by an analysis similar to what we did in birthday paradox, we can show that the probability of collision is small. However, the problem here is that to record the uniform hash function, we need $|\mathcal{U}| \log N$ bits, which is too big. Therefore, instead of choosing uniformly from all the possible mappings, we choose uniformly from a smaller set of functions. This motivates us to use hash functions with limited independence.

4.2 Limited Independence

Definition 4.1 (One-way Independence)**.** *Let $\mathcal{H}$ be a family of hash functions, we say $\mathcal{H}$ is one-way independent if for all $x_1 \in \mathcal{U}$ and for all $a_1 \in [N]$, we have*

$$\mathbb{P}_{h\sim H}[h(x_1) = a_1] = \frac{1}{N}.$$

Note that the above definition of one-way independence is not enough for a good family of hash functions. The family of constant functions $\{h_1, h_2, ..., h_N\}$ where $h_i(x) = i$ for every $x \in \mathcal{U}$ is one-way independent. Constant functions give us the largest amount of collisions we can imagine. However, when we take one step further and use a pairwise independent family of functions, we are able to achieve small collision probability.

Definition 4.2 (Pairwise Independence)**.** *Let $\mathcal{H}$ be a family of hash functions, we say $\mathcal{H}$ is pairwise independent if for all distinct $x_1, x_2 \in \mathcal{U}$ and for all $a_1, a_2 \in [N]$, we have*

$$\mathbb{P}_{h\sim H}[h(x_1) = a_1, h(x_2) = a_2] = \frac{1}{N^2}.$$

In fact, in the case of our problem, hash function families with the below definition of approximate pairwise independence property is sufficient.

Definition 4.3 (Approximate Pairwise Independence)**.** *Let $\mathcal{H}$ be a family of hash functions, we say $\mathcal{H}$ is α-approximate pairwise independent if for all distinct $x_1, x_2 \in \mathcal{U}$ and for all $a_1, a_2 \in [N]$, we have*

$$\mathop{\mathbb{P}}_{h\sim H}[h(x_1) = a_1, h(x_2) = a_2] \leq \frac{\alpha}{N^2}.$$

Before proving that pairwise independence is sufficient for a good family of hash functions, we remark that we can extend the definitions 4.1 and 4.2 to k-wise independence. Although pairwise independence is already sufficient for our application today, k-wise independent hash functions are very important objects in computer science, and thus have found a lot of applications elsewhere.

Definition 4.4 (k-wise Independence)**.** *Let $\mathcal{H}$ be a family of hash functions, we say $\mathcal{H}$ is k-wise independent if for all distinct $x_1, x_2, ...x_k \in \mathcal{U}$ and for all $a_1, a_2, ..., a_k \in [N]$, we have*

$$\mathop{\mathbb{P}}_{h\sim H}[h(x_1) = a_1, h(x_2) = a_2, ..., h(x_k) = a_k] = \frac{1}{N^k}.$$

4.3 Birthday Paradox Revisit

Now, let us revisit the analysis of the birthday paradox. We will see that the actual property that we need is approximate pairwise independence instead of mutual independence.

Similar to the analysis of the birthday paradox, we define Y_{ij} to be the indicator random variables that $h(X_i) = h(X_j)$, and let $Y = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} Y_{ij}$. Suppose that $\mathcal{H}$ is an α-approximate pairwise independent hash function family, then for every distinct $i, j \in [n]$

$$\mathbb{E}[Y_{ij}] = \mathop{\mathbb{P}}_{h\sim\mathcal{H}}[h(X_i) = h(X_j)] = \sum_{a\in[N]} \mathop{\mathbb{P}}_{h\sim\mathcal{H}}[h(X_i) = a, h(X_j) = a] \leq \frac{\alpha}{N^2}\cdot N = \frac{\alpha}{N}.$$

Therefore

$$\mathbb{E}[Y] = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \mathbb{E}[Y_{ij}] \leq \binom{n}{2}\alpha/N.$$

So if $n^2 < N/\alpha$, then by Markov's inequality, we have

$$\mathop{\mathbb{P}}_{h\sim\mathcal{H}}[\forall \text{ distinct } i, j \in \{1, ..., n\}, h(X_i) \neq h(X_j)] = \mathbb{P}[Y = 0] \geq \frac{1}{2}$$

The above analysis shows that a family of hash functions with the property of α-approximate pair independence for some constant α would be suffice for our purpose.

4.4 Double Hashing

The material of this section follows from the work of Fredman et al. []. As we will see in the next section, to store the the (approximate) pairwise independence hash functions, we will need $O(\log n)$ space. One plausible hashing scheme would be storing a linked list for each of the possible images of our hash function. If we use this hashing scheme directly, we will also need $O(n^2)$ space to store the starting nodes of the linked lists of the $O(n^2)$ possible images. Therefore, it requires $O(n^2)$ space overall to implement such a hashing

scheme, which is not affordable when n is too large. In fact, we can save more memory by using the following two layers hashing scheme.

Instead of choosing $N = \Theta(n^2)$, we choose $N = \Theta(n)$ and use our hashing algorithm to map the inputs to N buckets. We still maintain a link list for each of the buckets. For this choice of value of N, we should not expect no collision happen. However, we can bound the number of pairs of inputs elements that map to the same bucket. Then, by hashing elements in the same bucket in a second tier we avoid collision with a constant probability.

To analyze the above scheme, we use Z_i to denote the number of elements that map to the i-th location. Then, when we do the second layer of hashing, according to Section 4.3, we should map the elements in the i-th bucket to $[\alpha Z_i^2]$ location. On the other hand, by the proof of Section 4.3,

$$\mathbb{E}\sum_{i=1}^{N} Z_i^2 = 2\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \mathbb{E}Y_{ij} + n = \frac{2\alpha\binom{n}{2}}{N} + n = O(\alpha n).$$

Therefore, the expected amount of memory we will use is $O(n)$.

4.5 Construction of Pairwise Independent Hash Function

Let p be a prime so that $|U| \le p \le 2|U|$. Let variables a and b both be uniformly chosen from $\{0, ..., p-1\}$ such that $a \neq 0$. We show that the family of functions $f_{a,b}(x) = ax + b \mod p$ is pairwise independent. Note that these functions map $[p] \to [p]$. Later we see that using a mod operation we can construct pairwise independent hash functions that map $[p] \to [N]$.

Claim 4.5. *For all $x, y \in \{0, ..., p-1\}$ such that $x \neq y$, and for all $s, t \in \{0, ..., p-1\}$ such that $s \neq t$, we have*

$$\mathbb{P}_{a,b}[f_{a,b}(x) = s, f_{a,b}(y) = t] = \frac{1}{p(p-1)}$$

Proof. When $f_{a,b}(x) = s$ and $f_{a,b}(y) = t$, we have $ax + b \equiv s \mod p$, and $ay + b \equiv t \mod p$. Subtracting one from the other, we get

$$a(x - y) \equiv s - t \mod p.$$

Since p is a prime number, for every number $k \in \{1, ..., p-1\}$, there is a unique (modular) inverse k^{-1} of k so that $kk^{-1} \equiv 1 \mod p$. We do not discuss the algorithm for finding modular inverse, we refer students to https://en.wikipedia.org/wiki/Modular_multiplicative_inverse for details.

Since $x \neq y$, $x - y$ has a modular inverse. So, we can write solve the above system of modular equations for a and b; in particular, we have

$$a \equiv (s - t)(x - y)^{-1} \mod p.$$

Note that since $s \neq t$, we have $a \neq 0$ in above as desired. Furthermore,

$$b \equiv s - ax \mod p$$

The above analysis shows that for x, y, s, t given in the statement of the above claim, there exists a unique solution (a, b) so that $f_{a,b}(x) = s$ and $f_{a,b}(y) = t$. Since there are $p(p-1)$ possible values for (a, b) to take and (a, b) is chosen uniformly from them, the claim follows. □

Now, we choose the family of hash functions to be $\mathcal{H} = \{h_{a,b}\}$ where

$$h_{a,b}(x) = f_{a,b}(x) \mod N.$$

Note that to store this function in memory, we only have to store a and b, which takes only $O(\log p) = O(\log |U|)$ many bits. For the particular application to Hashing universe U, we can use another idea to reduce the memory size to $O(\log n)$. Please refere to [, Lem 2] for details.

Now, we show that $\mathcal{H}$ is the family of hash functions with 2-approximate pairwise independence property.

Claim 4.6. *For all $x, y \in \mathcal{U}$ so that $x \neq y$, we have*

$$\mathop{\mathbb{P}}_{a,b}[h_{a,b}(x) = h_{a,b}(y)] \leq \frac{1}{N}.$$

Proof. For all $x, y \in \mathcal{U}$ so that $x \neq y$, $h_{a,b}(x) = h_{a,b}(y)$ if and only if $f_{a,b}(x) \equiv f_{a,b}(y) \mod N$. Thus, by Claim 4.5

$$\begin{aligned}
\mathop{\mathbb{P}}_{a,b}[h_{a,b}(x) = h_{a,b}(y)] &= \mathop{\mathbb{P}}_{a,b}[f_{a,b}(x) \equiv f_{a,b}(y) \mod N] \\
&= \sum_{0 \leq s,t < p : s \neq t} \mathbb{P}[f_{a,b}(x) = s, f_{a,b}(y) = t] \, \mathbb{I}[s \equiv t \bmod N] \\
&= \sum_{0 \leq s,t < p : s \neq t} \frac{\mathbb{I}[s \equiv t \bmod N]}{p(p-1)} \\
&\leq \frac{1}{p(p-1)} \cdot \frac{p}{N} = \frac{1}{N}
\end{aligned}$$

The last inequality follows by the fact that for any $x \in [p]$ there are at most p/n numbers t such that $s \neq t$ and $s \equiv t \bmod N$. □

Note that the family of functions that we construct above is approximately pairwise independent but that is enough for all interesting applications.

We remark that we can extend the above construction and obtain a family of hash functions that is k-wise independent. For some prime number p, consider the family of hash functions

$$f_{a_0,\ldots,a_{k-1}}(x) = a_{k-1}x^{k-1} + \ldots + a_1 x + a_0,$$

where $a_0, \ldots, a_{k-1}$ are uniformly chosen in $\{0, 1, \ldots, p-1\}$. Similar to the invertible argument we used above, the proof that this construction is k-wise independent follows from the fact that the Vandermonde matrix

$$\begin{bmatrix}
1 & 1 & \cdots & 1 \\
x_1 & x_2 & \cdots & x_k \\
x_1^2 & x_2^2 & \cdots & x_k^2 \\
\vdots & \vdots & \ddots & \vdots \\
x_1^{k-1} & x_2^{k-1} & \cdots & x_k^{k-1}
\end{bmatrix}$$

is invertible for distinct $x_1, \ldots, x_k$. So, in general we can store a k-wise independent hash function with only $O(k \log |U|)$ amount of memory.

4.6 Introduction to Streaming Algorithms

As an application of hashing, we are going to discuss streaming algorithms in the next a couple of lectures. Streaming algorithms has become a hot topic in theoretical computer science nowadays because of the massive amount of data we have to process. Typically, we do not have enough space to store the entire data. Instead, we process the data in a streaming fashion, and sketch the information we want from the data by a few passes.

We will talk about algorithms for F_0 and F_2 estimation. Those are classic results appeared in the first paper of streaming algorithms []. The problem is as follows.

Let $\mathcal{U} = \{1, ..., |\mathcal{U}|\}$ be a large universe of numbers, and let $X_1, ..., X_n$ be a sequence of numbers in $\mathcal{U}$. Let $f_i = \sum_{j=1}^{n} \mathbb{I}[X_j = i]$ be the number of times i appears in the sequence. For $0 \leq k \leq \infty$, we let $F_k = \sum_{i=1}^{|\mathcal{U}|} f_i^k$, where we define $0^0 = 0$. The interesting values of k for us are

- When $k = 0$, F_0 counts the number of distinct elements in the sequence.
- When $k = 2$, F_2 is the second moment of the vector $(f_1, ..., f_{|\mathcal{U}|})$.
- When $k = \infty$, F_∞ corresponds to the number of times the most frequent number shows up in the sequence.

In the next lecture, we are going to prove the following theorem.

Theorem 4.7. *With $(1 - \delta)$ probability and using $O(\frac{\log |\mathcal{U}| + \log n}{\epsilon^2} \cdot \log \frac{1}{\delta})$ space, we can give a $(1 - \epsilon)$ approximation of F_0 and F_2 (in the worst case).*

We remark that allowing randomness and approximated solution is crucial to us. There is no hope to use a deterministic or exact algorithm to achieve logarithmic amount of space. Please see Tim Roughgarden's Lecture notes for more details.

References

[] M. L. Fredman, J. Komlós, and E. Szemerédi. "Storing a Sparse Table with 0(1) Worst Case Access Time". In: *J. ACM* 31.3 (June 1984), pp. 538–544 (cit. on pp. 4-2, 4-4).

[] N. Alon, Y. Matias, and M. Szegedy. "The space complexity of approximating the frequency moments". In: *STOC*. ACM. 1996, pp. 20–29 (cit. on p. 4-5).