CS 388R: Randomized Algorithms, Fall 2021 September 23th, 2021

Lecture 9: Limited Independence

Prof. Eric Price *Scribe: Dimitrios Christou, Konstantinos Stavropoulos*

NOTE: THESE NOTES HAVE NOT BEEN EDITED OR CHECKED FOR CORRECTNESS

1 Overview

Over the last lectures, we studied various ways of hashing and storing items (such as Balls and Bins, Power of 2 Choices, Cuckoo Hashing and Bloom Filters) while analyzing different quantities of interest such as insertion times, deletion times, query times and storage size. For all of these problems, we operated under the assumption that the hashing functions were uniformly random and neglected to account for the space that is required in order to store these hash functions. However, in order to write down a hash function that hashes elements in a set $[U]$, we would typically need $O(UlogU)$-bits which totally dominates the space guarantees we had achieved for the problems. Even if we end up storing $n < U$-items and write-down the hash values only for these, we would still need $O(nlogU)$ which is too large, creating an issue.

In order to address this problem, we will introduce the notion of "*Limited Independence*" and abandon the idea of random hash functions. We begin by listing the upsides and downsides of fully independent families of hash functions and compare them with the case of limited independence.

Fully Independent. This is also known as the Random Oracle model. The primary benefits of this model are:

1. Hash functions are free and their storage requirement don't have to be accounted for in the analysis.
2. Sometimes using this model is totally fine. For example, in Load Balancing we don't need to store any values and so we can get the randomness requirement by just flipping some coins.
3. Sometimes we can approximate this model. For example, in high-entropy inputs even if the hash functions are not random, the results end-up being approximately random. Also, in cryptographic settings assuming full independence is justified since verifications can be computationally hard problems that we believe to not be solvable.
4. Using this model allows us to establish better and easier bounds due to the convenience that independence provides.

Limited Independent. While all of the advantages of the Fully Independent model make it very useful, the truth is that we are not entirely honest in our analysis. This is solved by considering the limited independence model, but comes with the extra requirement of needing to pay for the space needed to store used hash functions. The benefits of this model can be summarized as:

1. We are really honest about the cost of using hash functions.
2. We are able to get theoretical guarantees for all inputs, even those that are adverserially created.
3. Per hash function, $O(k)$ words of space can give us k-wise independence, allowing for sets of size $\leq k$ to be treated as independent.

Still, this model is more strict and we will have to sacrifice some of the nice theoretical guaranteed that we achieved for the Random Oracle model. In this lecture, we will how we can analyze problems in this model. We will introduce the notions of "*Universal*" and "*Pairwise Independence*" hash families, show examples of hash functions that belong in these families and conclude with the description and analysis of "*Perfect Hashing*".

2 Definitions

Generally speaking, when designing a hashing system one of the quantities we wish to minimize is the expected look-up time of an item. Let $h : U \to [M]$ be our hash function that hashes n items. We can compute:

$$\mathbb{E}[\text{time to query } x] \leq 1 + \sum_{y \neq x} \mathbb{P}[h(x) = h(y)] \leq 1 + n \max_{y \neq x} \mathbb{P}[h(x) = h(y)]$$

If we assume full independence, then we have that for any $x, y \in U$ it holds $\mathbb{P}[h(x) = h(y)] = 1/M$. We would like to be able to get similar guarantees even when full independence doesn't hold. This motivates the following definitions:

Definition 1. *A hash family H of hash functions $h : U \to [M]$ is called **universal** if $\forall x, y \in U$ such that $x \neq y$ it holds*

$$\mathbb{P}_{h \in H}[h(x) = h(y)] \leq \frac{1}{M}$$

where the probability is taken over selecting one hash function $h \in H$ uniformly at random.

Definition 2. *A hash family H of hash functions $h : U \to [M]$ is called **approximately universal** if $\forall x, y \in U$ such that $x \neq y$ and some $\epsilon > 0$ it holds*

$$\mathbb{P}_{h \in H}[h(x) = h(y)] \leq \frac{1 + \epsilon}{M}$$

where the probability is taken over selecting one hash function $h \in H$ uniformly at random.

The notion of universality bounds the probability that any two items will collide after we hash them. However, some times we might need to examine larger sets of random variables. In these cases, an even stronger condition might be required.

Definition 3. *A hash family H of hash functions $h : U \to [M]$ is called **pairwise independent** if $\forall x, y \in U$ such that $x \neq y$ and $\forall i, j \in [M]$ it holds*

$$\mathbb{P}_{h \in H}[h(x) = i \cap h(y) = j] \leq \frac{1}{M^2}$$

where the probability is taken over selecting one hash function $h \in H$ uniformly at random.

Finally, the concept of pairwise independence can be generalized in the following straight-forward way:

Definition 4. *A hash family H of hash functions $h : U \to [M]$ is called* ***k-wise independent*** *if $\forall x_1, \ldots, x_k \in U$ such that $x_a \neq x_b \; \forall a, b \in [k] : a \neq b$ and $\forall i_1, \ldots, i_k \in [M]$ it holds*

$$\mathbb{P}_{h \in H}[h(x_j) = i_j \;\; \forall j \in [k]] \leq \frac{1}{M^k}$$

where the probability is taken over selecting one hash function $h \in H$ uniformly at random.

To understand the importance of these definitions, consider the case of throwing n balls at random into a collection of bins S. Let y_i be used to denotße the number of balls in bin i. Then, it is easy to see that:

$$y_i = \sum_{j \in [n]} 1[h(j) = i]$$

and thus

$$\mathbb{E}[\sum_{i \in S} y_i^k] = \mathbb{E}[\sum_{i \in S} (\sum_{j \in [n]} 1[h(j) = i])^k]$$

In that case, if we have k-wise independence we can proceed and analyze it as usual (when we were assuming full-independence). This shows that when computing k-th moments under k-wise independence, the analysis does not change from the fully independent model.

For example, in Cuckoo Hashing we proved that the maximum component size is $O(logn)$ in expectation, and thus having $logn$-wise independence allows us to use Cuckoo Hashing with the same guarantees. Similarly, having $O(\frac{logn}{loglogn})$-wise independence allows us to get the same theoretical guarantees in the Balls and Bins setting.

3 Implementations

The next step is to consider whether we can actually design families of hash functions that respect these constraints. We will show three different implementations.

3.1 Implementation 1 (Wegman-Carter)

This implementation was among the first universal functions and were considered in the original work of Carter and Wegman that introduced the notion of universality for hash function [**?**]. Let U be a finite field, for example the set of integers modulo P for some large prime P. Then, define

$$h_{a,b}(x) = (ax + b) \; mod \; P \; mod \; M$$

to be a hash functions from U to $[M]$. Then, the family of hash functions

$$H = \{h_{a,b} : a, b \in \{0, 1, \ldots, P - 1\}\}$$

is universal. Furthermore, the evaluation time is constant (fixed number of arithmetic operations) and in order to store the function, we only need to store parameters a, b when we create the table.

Note. This idea can also be applied to define a k-wise independent family, by considering

$$h(x) = (\sum_{i=0}^{k-1} a_i x^i) \; mod \; P \; mod \; M$$

3.2 Implementation 2

For this implementation, we will operate under the assumption that we are hashing u-bit inputs to m-bit words, that is $U = 2^u$ and $M = 2^m$ for some $u, m \in \mathbb{N}_+^*$. Notice that the assumption on U is not important because we can always increase our input universe to the next power of two. The assumption on M could lead to some complications, but due to the modulo operations that will happen it is not a big deal. For the rest of the section, we consider binary representations of the items and hash values.

We fill a matrix $A \in \{0,1\}^{m,u}$ with random bits and pick a random vector $b \in \{0,1\}^m$. Then, for any u-bit input x we define the mapping

$$h_{A,b}(x) = Ax + b$$

over the $\mathbb{F}_2$ field (which means that all operations are binary).

In order to evaluate this function we need $O(um \cdot 1 + m) = O(mu)$-operations and in order to store it, we only need to store the values of A and b which is equivalent to storing $O(mu)$-bits or $logM$ u-bit words.

Exercise. Prove that it suffices to consider only Toeplitz matrices A. In that case, we would only need to store $2u$-bits or 2 u-bit words for A.

Let H be used to denote the family of hash functions $h_{A,b}$ over binary matrices A and binary vectors b.

Theorem 5. *Family H is universal.*

Proof. Fix two u-bit words x, y such that $x \neq y$. Then:

$$\mathbb{P}_{A,b}[h_{A,b}(x) = h_{A,b}(y)] = \mathbb{P}_{A,b}[Ax + b = Ay + b] = \mathbb{P}_A[A(x - y) = 0]$$

Since the rows of A are independent, if a is used to denote any random binary row, we get

$$\mathbb{P}_{A,b}[h_{A,b}(x) = h_{A,b}(y)] = (\mathbb{P}_a[aw = 0])^m$$

where $w \neq 0$.

Let w ne any non-zero binary vector in $\{0,1\}^m$ and $v \in \{0,1\}^m$ be a uniformly random binary vector. Let $i \in [m]$ be such that $w_i = 1$ (it exists since $w \neq 0$). Then we get:

$$\mathbb{P}[v \cdot w = 0 \; mod \; 2] = \mathbb{P}[v_i = \sum_{j \neq i} v_j w_j \; mod \; 2] = \frac{1}{2}$$

since each coordinate of v is picked at random.

Using this observation, we immediately get that

$$\mathbb{P}_{A,b}[h_{A,b}(x) = h_{A,b}(y)] = \frac{1}{2^m} = \frac{1}{M}$$

which proves that H is universal. □

In fact, we can show an even stronger condition for H:

Theorem 6. *Family H is pairwise independent.*

Proof. Fix any two u-bit words x, y such that $x \neq y$ and any two m-bit words α, β. Then:

$$\begin{aligned}\mathbb{P}_{A,b}[h_{A,b}(x) = \alpha \cap h_{A,b}(y) = \beta] &= \mathbb{P}_{A,b}[Ax + b = \alpha \cap Ay + b = \beta] \\ &= \mathbb{P}_{A,b}[Ax + b = \alpha \cap A(y - x) = \beta - \alpha] \\ &= \mathbb{P}_{A}[A(y - x) = \beta - \alpha] \cdot \mathbb{P}_{A,b}[b = \alpha - Ax | A(y - x) = \beta - \alpha]\end{aligned}$$

since the second event does not depend on b. The first probability is similar to the one computed in the proof of universality, but with the difference that now Aw is equal to a fixed value and not 0. Using similar arguments, we can prove that this probability is also $\frac{1}{M}$. Thus:

$$\mathbb{P}_{A,b}[h_{A,b}(x) = \frac{1}{M}\mathbb{P}_{A,b}[b = \alpha - Ax | A(y - x) = \beta - \alpha]$$

Finally, if I fix A then $\alpha - Ax$ is a fixed value and b is uniformly picked from $\{0,1\}^M$, thus

$$\mathbb{P}_{A,b}[b = \alpha - Ax | A(y - x) = \beta - \alpha] = (\frac{1}{2})^m = \frac{1}{M}$$

Combining, we get that $\mathbb{P}_{A,b}[h_{A,b}(x) = \alpha \cap h_{A,b}(y) = \beta] = \frac{1}{M^2}$ which proves that H is pair-wise independent. □

As a final comment, we mention that pairwise independence implies universality, so the first proof was redundant.

3.3 Implementation 3

Finally, we present a family of hash functions that performs well in practice, since each hash function it contains is indexed by a single value in $[U - 1]$ and consists of a multiplication (we do not mind if we have an overflow, because we only care about the result modulo U) and a shift. However, the family is not universal, but it is 2-approximately universal.

In particular, we consider the family H that contains the hash functions of the form $h_a : [U] \to [M]$ (where $U = 2^u$ and $M = 2^m$) such that a is odd in $\{1, 2, \ldots, U - 1\}$ and

$$h_a(x) := (ax \mod U) >> u - m\,,$$

or in other words, we multiply x with a and keep the top m bits of the result, ignoring any possible overflown bits of ax when it is viewed as a u-bit word.

The first observation is that it is essential to only include odd values for a, because for any even a, the words $x_1 = 0$ and $x_2 = u/2$ are hashed to the same value (0) and hence the probability that x_1 and x_2 collide increases for any even even value we enable (if we enabled all the even values, then the probability of x_1 and x_2 colliding would be at least $1/2$).

Example. Consider $x = 2^{u-1}$. Then, for any (odd) a, $h_a(x) = M/2$, because

$$ax = 2kx + x = x \mod U,$$

since $2x \mod U = 0$ and the first m bits of x are $(10\ldots0)_2 = 2^{m-1} = M/2$.

Claim. $H := \{h_a : a \text{ odd}\}$ is 2-approximately universal.

Intuitively, the above claim is true because the top bits of $a(x-y) \mod U$ are fully randomized, except possibly from the m-th most significant bit which for any $x-y$ ending in $(10\ldots0)_2$ ($u-m+1$ least significant bits) remains 1 regardless of the choice of a (if there were more zeros in the end, then $h_a(x) \neq h_a(y)$, because $a(x-y) \mod U$ would have both a 1 and some 0 bits).

4 Perfect Hashing

We say that $h : [U] \to [M]$ is "perfect" for S if $h(x) \neq h(y)$ for any $x, y \in S$ with $x \neq y$. But how can we construct perfect hash functions for $|S| = n$?

We have that the expected number of collisions for a universal family of hash functions H is upper bounded by

$$\binom{n}{2} \cdot \frac{1}{M} < \frac{n^2}{M}.$$

Therefore, if we pick $M = 10n^2$, then we have zero collisions with probability at least 90% (by Markov's Inequality).

However, applying this idea directly would require $O(n^2)$ space. We will show that there is a way to construct perfect hash functions with expected constant time per insertion using $O(n)$ space.

The idea is to first use a hash table that is pairwise independent and has n bins. Let y_i be the load in bin i. For each bin, we create a hash table of size $O(y_i^2)$ (which will be perfect with constant probability). Hence, we have 2 levels of hashing.

The total space we use is $O(\sum_i y_i^2)$. We have that

$$\mathbb{E}[\sum_i y_i^2] = \sum_i \mathbb{E}[y_i^2] = \sum_i \mathbb{E}[2\binom{y_i}{2} + y_i] = 2\sum_i \mathbb{E}[\#\text{collisions in bin } i] + \mathbb{E}[\text{total } \# \text{ collisions}] \leq$$

$$\leq \frac{n^2}{M} + n = O(n)\,.$$

However, in the above construction, some probabilities of failure are involved (e.g. the hash table corresponding to bin i could be non-perfect). But, since we can verify whether our construction is acceptable, we can create a Las Vegas algorithm for this problem, via "rejection sampling". In particular:

1. Pick random global $h : [U] \to [n]$
    - Compute y_i.
    - If $\sum_i y_i^2 \geq 10n$, go back to step 1.
2. For each cell of the hash table:
    - Pick $h_i : U \to [10y_i^2]$.
    - If collisions exist (non-perfect), then repeat.

The above algorithm runs in expected $O(n)$ time, which is $O(1)$ per insertion and creates a perfect hash function that uses $O(n)$ space.

References

[CW79] J.Lawrence Carter, Mark N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143–154, 1979.