Introduction to pairwise independent hashing

Recall the problem of hash tables: Let N be a large number and let $S \subset \{1,\ldots,N\}$ be a set such that $|S| \ll N$. We wish to store the elements of S in a table of size $\approx |S|$, such that we can efficiently check whether a number $x \in \{1,\ldots,N\}$ is in S or not. A common solution to this problem is using "hash tables": Let M denote the size of the table in which we want to store S. We choose a function $h : \{1,\ldots,N\} \to \{1,\ldots,M\}$. Then, for every $1 \leq i \leq M$, we store in the i-th cell of the table all the elements $x \in S$ such that $h(x) = i$. Whenever we are required to check whether a number x is in S, we read the $h(x)$-th cell of the table and check if it contains x. If each cell in the table contains only a few elements of S, then this solution is very efficient.

How do we make sure that each cell in the table does not contain too many elements? We need to choose h such that there are not many "collisions", i.e., there are not many pairs of elements $x, y \in S$ such that $h(x) = h(y)$. In order to achieve it, we choose a function h that maps the elements of $\{1,\ldots,N\}$ to $\{1,\ldots,M\}$ in a "random-like" manner. If h is "sufficiently random" and if M is sufficiently large comparing to S, then we indeed expect that there won't be many collisions. The reason is that, if we would have chosen $h(x)$ and $h(y)$ completely at random, then the probability that $h(x) = h(y)$ would have been $\frac{1}{M}$, which is small. The problem is to choose a function h that has such a random property.

One possible solution for choosing such a function h is to choose h to be a random function from $\{1,\ldots,N\}$ to $\{1,\ldots,M\}$. The problem with this solution is that in order to compute h later, we will need to store in the memory a table that says for every $x \in \{1,\ldots,N\}$ what is $h(x)$, and storing such a table will require too much memory. We will have to find another solution.

Our problem can be now stated as follows: We need to find a function h that *behaves like a random function*, even though *it is not really a random function*. Furthermore, we want to be able to compute h without using too much memory. This problem is very, very common in Theoretical Computer Science, and arises in many contexts rather than just in the context of hash tables - In fact, this is one of the major problems of the theory of Derandomization.

In order to solve this problem, we first need to define what does it mean for a function h to "behave like a random function". In the theory of Derandomization, there are many possible definitions of the "behavior of a random function", and each of those definitions is useful in other settings. In this course, we will only need one of those definitions, known as "pairwise independence".

Before introducing this definition, we first note that "pairwise independence" is not a property of a *single function*, but rather a property of a *family of functions*. A family of functions is said to be pairwise independent if when we choose a random function *from this family*, then the distrubition of the images of the function is "somewhat random". The formal definition is as follows:

Definition. Let H be a family of functions from $\{1,\ldots,N\}$ to $\{1,\ldots,M\}$. The family H is said to be pairwise independent if for every $x, y \in \{1,\ldots,N\}$ such that $x \neq y$, and for every $a, b \in \{1,\ldots,M\}$ it holds that

$$\Pr_{h \in H}[h(x) = a \wedge h(y) = b] = \frac{1}{M^2}$$

That is, if h is a function chosen uniformly at random from H, then the random variables $h(x)$ and $h(y)$ are *uniformly distributed* and *pairwise independent*.

Suppose that we have such a pairwise independent family H, such that every function in H can be represented using a small amount of bits (say, $O(\log N)$) and such that every function in H can be computed efficiently. Using such a family, we can solve the problem of hash tables we described above. If we choose a random function $h \in H$, then for every two elements $x, y \in S$, the probability that $h(x)$ and $h(y)$ will collide

is $\frac{1}{M}$, and the expected number of collisions will be

$$\sum_{x,y \in S} \frac{1}{M} = \binom{|S|}{2} / M$$

Therefore, if we choose $M = |S|^2$, the expected number of collisions will be smaller than 1, which is what we want. This solution is very useful in applications where $N \gg |S|^2$.

In this course, we will often use efficiently computable pairwise independent families of functions. Usually, we will use functions that map $\{0,1\}^n$ to $\{0,1\}^m$, rather than $\{1,\ldots,N\}$ to $\{1,\ldots,M\}$. We will denote such a family by H_m^n. That is:

Definition. For every $n, m \in \mathbb{N}$, the set H_m^n is a family of functions from $\{0,1\}^n$ to $\{0,1\}^m$ that satisfies the following properties:

1. For every $x, y \in \{0,1\}^n$ such that $x \neq y$ and for every $a, b \in \{0,1\}^m$, it holds that

$$\Pr_{h \in H_m^n}[h(x) = a \wedge h(y) = b] = 2^{-2m}$$

2. Every function $h \in H_m^n$ can be represented using a string of $O(n+m)$ bits. This string is called the description of h. Actually, for the applications we present in this course, it will suffice to have descriptions of length poly (n, m), rather than $O(n+m)$.

3. Each function $h \in H_m^n$ can be efficiently computed given its description. That is, there is a polynomial time algorithm that, given the description of a function $h \in H_m^n$ and a string $x \in \{0,1\}^n$, computes $h(x)$.

Of course, before we can use such families, we need to prove that they exist. Such families can indeed be constructed, but we won't show it here. The description of such constructions can be found in Appendix D.2.2 of the book.

We finish this introduction by describing a useful property of pairwise independent families of functions, called "Mixing". Fix some set $S \subseteq \{0,1\}^n$ and a subset $T \subseteq \{0,1\}^m$. Suppose that we choose a random function $h : \{0,1\}^n \to \{0,1\}^m$ (not necessarily in H_m^n). How many elements $x \in S$ satisfy $h(x) \in T$? Well, since for every $x \in S$ and $a \in T$ there is a probability of 2^{-m} that $h(x) = a$, we would expect that about $|S| \cdot |T| / 2^m$.

Now, note that this argument should also hold when h is a random function in H_m^n, rather than any random function, because by the definition of H_m^n, it also holds that for a random function $h \in H_m^n$, for every $x \in S$ and for every $a \in T$, the probability that $h(x) = a$ is 2^{-m}. Indeed, the mixing property of H_m^n says that with high probability, the number of elements of S that map into T is $|S| \cdot |T| / 2^m$, up to a small error we denote by ε. Formally, this property is stated as follows:

Lemma (Mixing). *Let $S \subseteq \{0,1\}^n$, $T \subseteq \{0,1\}^m$, and let h be function in H_m^n chosen uniformly at random. Then, for every $\varepsilon > 0$ it holds that*

$$(1-\varepsilon)\frac{|S| \cdot |T|}{2^m} < |\{x \in S : h(x) \in T\}| < (1+\varepsilon)\frac{|S| \cdot |T|}{2^m}$$

with probability of at least $1 - \frac{2^m}{\varepsilon^2 \cdot |S| \cdot |T|}$.

The Mixing lemma can be proved quite easily using Chebyshev inequality. A useful special case of the mixing lemma is when $|T| = 1$. This special case means that for every $a \in \{0,1\}^m$, it holds that $\left|h^{-1}(a)\right| \approx |S| / 2^m$ with high probability.