# 1 Hashing

Hashing functions are an integral part of minwise hashing, as it uses the values of hashing functions to determine similarity between two sets. Therefore, it is necessary that we have a small insight into universal hash functions before moving on to minwise hashing in the next section.

## 1.1 Introduction to hash functions

Imagine a universe of keys $U = \{u_0, u_1, \ldots, u_{n-1}\}$ and a range $[r] = \{0, 1, \ldots, m-1\}$. Given a hash function $h : U \to [r]$ which takes any $u_i, i = 0, 1, \ldots, n-1$ as argument, it should hold that $\forall u_i, \exists r_j \in [r]$ such that $h(u_i) = r_j$. What remains is to consider collisions, which we define as

$$\delta_h(x, y) = \begin{cases} 1 & \text{if } x \neq y \text{ and } h(x) = h(y) \\ 0 & \text{else} \end{cases} \tag{1}$$

for a given hashfunction $h$ and two keys $x$ and $y$. The goal of a hashing algorithm is then to minimize the number of collisions across all possible keys. A perfect hash function can assure that there are no collision at all. Unfortunately, to implement such a function would require at least $|U| \log_2 m$ bits [**?**], defeating the purpose of hash functions altogether. Fixed hashing algorithms have attempted to solve this problem. Unfortunately, their dependence on input cause a worst case average retrieval time of $\Theta(n)$ [**?**].

Universal hashing can circumvent the memory and computation cost of both random- and fixed hashing, without losing much precision. An introduction to universal hashing will follow, alongside two applications of said hashing which will be tested later in this paper.

## 1.2 Universal hash functions

The first mention of universal hashing was in [**?**], in which they define universality of hash functions as follows:

Given a class of hash functions $H = \{h : U \to [r]\}$, $H$ is said to be universal if $\forall x \forall y \in U$

$$\delta_H(x, y) \leq \frac{|H|}{m}$$

where, with $S \subset U$

$$\delta_H(x, S) = \sum_{h \in H} \sum_{y \in S} \delta_h(x, y)$$

That is, $H$ is said to be universal if

$$\Pr_h\left(h(x) = h(y)\right) \leq \frac{1}{m} \tag{2}$$

for a random $h \in H$. In many applications, $\Pr_h[h(x) = h(y)] \leq c/m$ for $c = O(1)$ is sufficiently low.

### 1.2.1 Carter and Wegman

One of the proposed universal hash functions in [**?**] is very easily applicable. Its definition goes as: given a prime $p \geq m$ and a hash function $h^C_{a,b} : U \to [r]$,

$$h^C_{a,b}(x) = ((a \cdot x + b) \mod p) \mod m \tag{3}$$

where $a$ and $b$ are integers mod $m$, with $a \neq 0$. We want to prove that $h^C_{a,b}(x)$ satisfies Eq. 2; thus proving that it is universal.

Let $x$ and $y$ be two randomly selected keys in $U$ where $x \neq y$. For a given hash function $h^C_{a,b}$,

$$r = a \cdot x + b \mod p$$

$$q = a \cdot y + b \mod p$$

If we subtract $r$ and $q$ from each other, we get

$$(r - q) \equiv a(x - y) \mod p$$

We see that $(r-q) > 0$ since the right-hand side (RHS) will always be a positive number by the following logic: For the RHS to equal zero, either $a(x - y) = 0$ or $a(x - y) = p$. It is impossible for $a(x - y)$ to equal zero, since $a$ is a positive number, and $x \neq y$. The only condition in which $a(x-y)$ could equal the prime number were if $a = p$ and $(x - y) = 1$ which cannot hold, since $a < m < p$. By this proof it has been shown that all Carter Wegman hash functions $\forall a \forall b, h^C_{a,b}$ will map to distinct values for the given $x$ and $y$, at least at the mod $p$ level.

### 1.2.2 Multiply-shift

This state-of-the-art scheme described in [**?**] reduces computation time by eliminating the need for the **mod** operator. This is especially useful when the key is larger than 32 bits, in which case Carter and Wegman's suggestion is quite costly [**?**].

Take a universe $U \geq 2^k$ which is all $k$-bit numbers. For $l = \{1, .., k\}$, the hash functions $h^D_a(x) : \{0, \ldots, 2^k - 1\} \to \{0, \ldots, 2^l - 1\}$ are then defined as

$$h^D_a(x) = (a \cdot x \mod 2^k)/2^{k-l} \tag{4}$$

for a random odd number $0 < a < 2^k$. $l$ is bitsize of the value the keys map to. The following C-like code shows just how simple the implementation of such an algorithm is

```
h(x)=(unsigned) (a*x) >> (k-l)
```

This scheme only nearly satisfies Eq. 2, as for two distinct $x, y \in U$ and any allowed $a$

$$\Pr_{h^D_a}[h^D_a(x) = h^D_a(y)] \leq \frac{1}{2^{l-1}} = \frac{2}{m} \tag{5}$$

If Eq. 5 is not sufficently precise, Wölfel [**?**, p.18-19] modified this scheme so that it met the requirement in Eq. 2. The hash function is then

$$h^D_{a,b} = ((a \cdot x + b) \mod 2^k) \div 2^{k-l}$$

where $a < 2^k$ is a positive odd number, and $0 \leq b < 2^{k-l}$. This way Eq. 2 is met for $x \neq y$. For a proof of this, consult [**?**]. The C-like implementation shown below reveals that the modifications are only minimal

```
h(x)=(unsigned)((a*x) + b) >> (k-l)
```