1 Hash functions

The goal of hash functions is to map elements from a large domain to a small one. Typically, to obtain the required guarantees, we would need not just one function, but a family of functions, where we would use randomness to sample a hash function from this family. Let \( H = \{ h : U \rightarrow R \} \) be a family of functions, mapping elements from a (large) universe to a (small) range. Ideally, we would like to obtain certain “random-like” properties of this family, while keeping its size small. This will be important for applications in data structures and streaming algorithms, where we would need to keep a “seed” which tells us which hash function we chose, as well as in de-randomization, where we can replace true random with pseudo-randomness, obtained via enumerating all the hash functions in a family.

We start with describing one of the more basic but very useful properties we can require from hash functions, that of being pairwise-independent.

**Definition 1.1** (Pairwise independent hash functions). A family \( H = \{ h : U \rightarrow R \} \) is said to be pairwise independent, if for any two distinct elements \( x_1 \neq x_2 \in U \), and any two (possibly equal) values \( y_1, y_2 \in R \),

\[
\Pr_{h \in H} [h(x_1) = y_1 \text{ and } h(x_2) = y_2] = \frac{1}{|R|^2}.
\]

2 Pairwise independent bits

We will construct a family of hash functions \( \mathcal{H} = \{ h : U \rightarrow R \} \) where \( R = \{0,1\} \). We will assume that \(|U| = 2^k\), by possible increasing the universe size, and identify \( U = \{0,1\}^k \). Define the following family of hash functions:

\[
\mathcal{H} = \{ h_{a,b}(x) = \langle a, x \rangle + b \pmod{2} : a \in \{0,1\}^k, b \in \{0,1\} \}.
\]

It is easy to see that \(|\mathcal{H}| = 2^{k+1} = 2|U|\). In order to show that \( \mathcal{H} \) is pairwise independent, we would need the following simple claim.
Claim 2.1. Let $x \in \{0, 1\}^k$, $x \neq 0$. Then

$$\Pr_{a \in \{0, 1\}^k} [\langle a, x \rangle \pmod{2} = 0] = \frac{1}{2}.$$  

Proof. Assume $x_i = 1$ for some $i \in [k]$. Then

$$\Pr_{a \in \{0, 1\}^k} [\langle a, x \rangle \pmod{2} = 0] = \Pr [a_i = \sum_{j \neq i} a_j x_j \pmod{2}] = \frac{1}{2}.$$  

\[ \square \]

Lemma 2.2. $\mathcal{H}$ as described above is pairwise independent.

Proof. Fix $x_1 \neq x_2 \in \{0, 1\}^k$ and $y_1, y_2 \in \{0, 1\}$. All the calculations below of $\langle a, x \rangle + b$ are modulo 2. We need to prove

$$\Pr_{a \in \{0, 1\}^k, b \in \{0, 1\}} [\langle a, x_1 \rangle + b = y_1 \text{ and } \langle a, x_2 \rangle + b = y_2] = \frac{1}{4}.$$  

If we just randomized over $a$ then by the claim, then for any $y \in \{0, 1\}$ by the claim,

$$\Pr_a [\langle a, x_1 \rangle + \langle a, x_2 \rangle = y] = \Pr_a [\langle a, x_1 \rangle = y] = \frac{1}{2}.$$  

Randomizing also over $b$ gives us the desired result.

$$\Pr_{a, b} [\langle a, x_1 \rangle + b = y_1 \text{ and } \langle a, x_2 \rangle + b = y_2] = \Pr_{a, b} [\langle a, x_1 \rangle + \langle a, x_2 \rangle = y_1 \oplus y_2 \text{ and } b = \langle a, x_1 \rangle + y_1] = \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}.$$  

\[ \square \]

An equivalent way to view this is as follows. We can generate a joint distribution over $n = |U|$ bits, such that any pair of bits is uniform. Let $U = \{u_1, \ldots, u_n\}$. To generate a random binary string $x_1, \ldots, x_n$, we sample $h \in \mathcal{H}$ uniformly and set $x_i = h(u_i)$. Our distribution has support of size at most $|\mathcal{H}| = O(n)$. That is, only $O(n)$ binary strings are possible. This is much fewer than the uniform distribution over $n$ bits, which is supported on all $2^n$ binary strings. In particular, we can represent a string by specifying the hash function which generated it, which only takes $\log |\mathcal{H}| = \log n + O(1)$ bits.

Example 2.3. For $n = 4$, we get that sampling a uniform string from the following set of $8 = 2^3$ strings is pairwise independent:

$$\{0000, 0011, 0101, 0110, 1001, 1010, 1100, 1111\}.$$  

2
2.1 Application: de-randomized MAXCUT

Let $G = (V, E)$ be a simple undirected graph. For $S \subset V$ let $E(S, S^c) = \{(u, v) \in E : u \in S, v \in S^c\}$ be the number of edges which cross the cut $S$. The MAXCUT problem asks to find the maximal number of edges in a cut.

$$\text{MAXCUT}(G) = \max_{S \subset V} |E(S, S^c)|$$

Computing the MAXCUT of a graph is known to be NP-hard. Still, there is a simple randomized algorithm which approximates it within factor 2. Let $n = |V|$ and $|V| = \{v_1, \ldots, v_n\}$.

**Lemma 2.4.** Let $x_1, \ldots, x_n \in \{0, 1\}$ be uniformly and independently chosen. Set $S = \{v_i : x_i = 1\}$. Then

$$\mathbb{E}_S [|E(S, S^c)|] \geq \frac{|E(G)|}{2} \geq \frac{\text{MAXCUT}(G)}{2}.$$  

**Proof.** For any choice of $S$ we have

$$|E(S, S^c)| = \sum_{(v_i, v_j) \in E} 1_{v_i \in S} 1_{v_j \in S^c}$$

Note that every undirected edge $\{u, v\}$ in $G$ is actually counted twice in the calculation above, once as $(u, v)$ and once as $(v, u)$. However, clearly at most one of these is in $E(S, S^c)$.

By linearity of expectation, the expected size of the cut is

$$\mathbb{E}_S [|E(S, S^c)|] = \sum_{(v_i, v_j) \in E} \mathbb{E}[1_{v_i \in S} 1_{v_j \notin S}] = \sum_{(v_i, v_j) \in E} \mathbb{E}[1_{x_i = 1} 1_{x_j = 0}] = \sum_{(v_i, v_j) \in E} \mathbb{P}[x_i = 1 \text{ and } x_j = 0] = 2|E(G)| \cdot \frac{1}{4} = \frac{|E(G)|}{2}.$$  

This implies that a random choice of $S$ has a non-negligible probability of giving a 2-approximation.

**Corollary 2.5.** $\Pr_S \left[ |E(S, S^c)| \geq \frac{|E(G)|}{2} \right] \geq \frac{1}{2|E(G)|} \geq \frac{1}{n^2}.$

**Proof.** Let $X = |E(S, S^c)|$ be a random variable counting the number of edges in a random cut. Let $\mu = |E(G)|/2$, where we know that $\mathbb{E}[X] \geq \mu$. Note that whenever $X < \mu$, we in fact have that $X \leq \mu - 1/2$, since $X$ is an integer and $\mu$ a half-integer. Also, note that always $X \leq |E(G)| \leq 2\mu$. Let $p = \Pr[X \geq \mu]$. Then

$$\mathbb{E}[X] = \mathbb{E}[X | X \geq \mu] \Pr[X \geq \mu] + \mathbb{E}[X | X \leq \mu - 1/2] \Pr[X \leq \mu - 1/2] \leq 2\mu \cdot p + (\mu - 1/2) \cdot (1 - p) \leq \mu - 1/2 + 2\mu p.$$  

So we must have $2\mu p \geq 1/2$, which means that $p \geq 1/(4\mu) \geq 1/(2|E(G)|)$.  

So in particular, we can sample $O(n^2)$ sets $S$, compute for each one its cut size, and we are guaranteed that with high probability, the maximum will be at least $|E(G)|/2$. We can derandomize randomized algorithm this using pairwise independent bits. As a side benefit, it will reduce the computation time from testing $O(n^2)$ sets to testing only $O(n)$ sets.

**Lemma 2.6.** Let $x_1, \ldots, x_n \in \{0, 1\}$ be pairwise independent bits. Set

$$S = \{v_i : x_i = 1\}.$$  

Then

$$\mathbb{E}_S[|E(S, S^c)|] \geq \frac{|E(G)|}{2}.$$  

**Proof.** The only place where we used the fact that the bits were uniform, where in the calculation that

$$\Pr[x_i = 1 \text{ and } x_j = 0] = \frac{1}{4}$$

for all distinct $i, j$. However, this is also true for pairwise independent bits. \square

In particular, for one of the $O(n)$ sets $S$ that we generate in the algorithm, we must have that $|E(S, S^c)|$ exceeds the average, and hence $|E(S, S^c)| \geq |E(G)|/2$.

### 2.2 Optimal sample size for pairwise independent bits

The previous application showed the usefulness of having small sample spaces for pairwise independent bits. We saw that we can generate $O(n)$ binary strings of length $n$, such that choosing one of them uniformly gives us pairwise independent bits. We next show that this is optimal.

**Lemma 2.7.** Let $X \subset \{0, 1\}^n$. Assume that for any distinct $i, j \in [n]$ and any $b', b'' \in \{0, 1\}$ we have

$$\Pr_{x \in X}[x_i = b' \text{ and } x_j = b''] = \frac{1}{4}.$$  

Then $|X| \geq n$.

**Proof.** Let $m = |X|$ and suppose $X = \{x^1, \ldots, x^m\}$. For any $i \in [n]$, construct a real vector $v_i \in \{-1, 1\}^m$ as follows:

$$(v_i)_t = (-1)^{(x^t)_i}.$$  

We will show that the set of vectors $\{v_1, \ldots, v_m\}$ are linearly independent over the reals, and hence span a subspace of dimension $n$. Hence, $m \geq n$. To do that, we first show that $\langle v_i, v_j \rangle = 0$ for all $i \neq j$. To see that, note that

$$\langle v_i, v_j \rangle = \sum_{x \in X} (-1)^{x_i}(-1)^{x_j} = \sum_{x \in X} (-1)^{x_i+x_j} = |\{x \in X : x_i = x_j\}| - |\{x \in X : x_i \neq x_j\}|.$$
By our assumption however, $|\{x \in X : x_i = x_j\}| = |\{x \in X : x_i \neq x_j\}| = 1/2$ and hence $\langle v_i, v_j \rangle = 0$. Now, assume towards contradiction that $v_1, \ldots, v_n$ are linearly independent. Then there exist coefficients $\alpha_1, \ldots, \alpha_n \in \mathbb{R}$, not all zero, such that

$$\sum \alpha_i v_i = 0.$$ 

However, for any $j \in [n]$, we have

$$0 = \left\langle \sum \alpha_i v_i, v_j \right\rangle = \sum \alpha_i \langle v_i, v_j \rangle = \alpha_j \|v_j\|^2 = |X| \alpha_j.$$

So we must have $\alpha_j = 0$ for all $j$, a contradiction. 

3 Hash functions with large ranges

We now consider the problem of constructing a family of hash functions $H = \{h : U \to R\}$ for large $R$. For simplicity, we will assume that $|R|$ is prime, although this requirement can be somewhat removed. So, let's identify $R = \mathbb{F}_p$ for a prime $p$. We may assume that $|U| = p^k$, by possibly increase the size of the universe by $p$. So, we can identify $U = \mathbb{F}_p^k$. Define the following family of hash functions:

$$H = \{h_{a,b}(x) = \langle a, x \rangle + b : a \in \mathbb{F}_p^k, b \in \mathbb{F}_p\}.$$ 

Note that $|H| = p^{k+1} = |U| \cdot |R|$. In order to show that $H$ is pairwise independent, we need the following generalized claim.

**Claim 3.1.** Let $x \in \mathbb{F}_p^k, x \neq 0$. Then for any $y \in \mathbb{F}_p$,

$$\Pr_{a \in \mathbb{F}_p^k} [\langle a, x \rangle = y] = \frac{1}{p}.$$ 

**Proof.** Assume $x_i \neq 0$ for some $i \in [k]$. Then

$$\Pr_{a \in \mathbb{F}_p^k} [\langle a, x \rangle = y] = \Pr \left[ a_i x_i = y - \sum_{j \neq i} a_j x_j \right].$$

Now, for every fixing of $\{a_j : j \neq i\}$, we have that $a_i x_i$ is uniformly distributed in $\mathbb{F}_p$, hence the probability that it equals any value is exactly $1/p$. 

**Lemma 3.2.** $H$ as described above is pairwise independent.

**Proof of lemma.** Fix $x_1 \neq x_2 \in \mathbb{F}_p^k$ and $y_1, y_2 \in \mathbb{F}_p$. All the calculations below of $\langle a, x \rangle + b$ are in $\mathbb{F}_p$. We need to prove

$$\Pr_{a \in \mathbb{F}_p^k, b \in \mathbb{F}_p} [\langle a, x_1 \rangle + b = y_1 \text{ and } \langle a, x_2 \rangle + b = y_2] = \frac{1}{p^2}.$$
If we just randomized over \(a\) then by the claim, then for any \(y \in \mathbb{F}_p\) by the claim,
\[
\Pr_a[\langle a, x_1 \rangle - \langle a, x_2 \rangle = y] = \Pr_a[\langle a, x_1 \rangle - x_2 = y] = \frac{1}{p}.
\]

Randomizing also over \(b\) gives us the desired result.
\[
\Pr_{a,b}[\langle a, x_1 \rangle + b = y_1 \text{ and } \langle a, x_2 \rangle + b = y_2] = \Pr_{a,b}[\langle a, x_1 \rangle - \langle a, x_2 \rangle = y_1 - y_2 \text{ and } b = \langle a, x_1 \rangle + y_1] = \frac{1}{p} \cdot \frac{1}{p} = \frac{1}{p^2}.
\]

\(\square\)

### 3.1 Application: collision free hashing

Let \(S \subseteq U\) be a set of objects. A hash function \(h : U \to R\) is said to be collision free for \(S\) if it is injective on \(S\). That is, \(h(x) \neq h(y)\) for all distinct \(x, y \in S\). We will show that if \(R\) is large enough, then any pairwise independent hash family contains many collision free hash functions for any small set \(S\). This is extremely useful: it allows to give lossless compression of elements from a large universe to a small range.

**Lemma 3.3.** Let \(\mathcal{H} : \{h : U \to R\}\) be a pairwise independent hash family. Let \(S \subseteq U\) be a set of size \(|S|^2 \leq |R|\). Then
\[
\Pr_{h \in \mathcal{H}}[h \text{ is collision free for } S] \geq \frac{1}{2}.
\]

**Proof.** Let \(h \in \mathcal{H}\) be uniformly chosen, and let \(X\) be a random variable that counts the number of collisions in \(S\). That is,
\[
X = \sum_{x \neq y \in S} 1_{h(x) = h(y)}.
\]

The expected value of \(X\) is
\[
E[X] = \sum_{x \neq y \in S} \Pr_{h \in \mathcal{H}}[h(x) = h(y)] = \left(\frac{|S|}{2}\right) \frac{1}{|R|} \leq \frac{|S|^2}{2|R|} \leq \frac{1}{2}.
\]

Let \(p = \Pr[X \geq 1]\) be the probability that there is at least one collision. Then
\[
E[X] = E[X|X = 0] \Pr[X = 0] + E[X|X \geq 1] \Pr[X \geq 1] \geq p.
\]

So, \(p \leq 1/2\), and hence at least half the functions \(h \in \mathcal{H}\) are collision free for \(S\). \(\square\)
We now show how to use pairwise independent hash functions, in order to design efficient dictionaries. Fix a universe $U$. For simplicity, we will assume that for any $R$ we have a family of pairwise independent hash functions $\mathcal{H} = \{h : U \rightarrow R\}$, and note that while our previous constructions required $R$ to be prime (or in fact, a prime power), this will at most double the size of the range, which at the end will only change our space requirements by a constant factor.

Given a set $S \subset U$ of size $|S| = n$, we would like to design a data structure which supports queries of the form “is $x \in S$?” Our goal will be to do so, while minimizing both the space requirements and the time it takes to answer a query. If we simply store the set as a list of $n$ elements, this takes space $O(n \log |U|)$, and queries take time $O(n \log |U|)$. We will see that this can be improved via hashing.

First, consider the following simple hashing scheme. Fix a range $R = \{1, \ldots, n^2\}$. Let $\mathcal{H} = \{h : U \rightarrow \mathbb{Z}_{n^2}\}$ be a pairwise independent hash function. We showed that a randomly chosen $h \in \mathcal{H}$ will be collisions free on $S$ with probability at least $1/2$. So, we can sample $h \in \mathcal{H}$ until we find such an $h$, and we will find one on average after two samples. Let $A$ be an array of length $n^2$. It will be mostly empty, except that we set $A[h(x)] = x$ for all $x \in S$.

Now, to check whether $x \in S$, we compute $h(x)$ and check whether $A[h(x)] = x$ or not. Thus, the query time is only $O(\log |U|)$. However, the space requirements are big: to store $n$ elements, we maintain an array of size $n^2$, which requires at least $n^2$ bits (and maybe even $O(n^2 \log |U|)$, depends on how clever is your implementation for storing the empty cells).

We now describe a two-step hashing scheme due to Ajtai, Komlos and Szemerédi which avoids this large waste of space. It will use only $O(n \log n + \log |U|)$ space, but would still allow for query time of $O(\log |U|)$. As a preliminary step, we apply the collision free hash scheme we just described. So, will assume from now on that $U = O(n^2)$ and that $S \subset U$ of size $|S| = n$.

**Step 1.** We first find a hash function $h : U \rightarrow [n]$ which has only $n$ collisions. Define

$$\text{Coll}(h, S) = |\{x, y \in S : x \neq y, h(x) = h(y)\}|.$$

If $\mathcal{H} = \{h : U \rightarrow [n]\}$ is a family of pairwise independent hash functions, then

$$\mathbb{E}_{h \in \mathcal{H}}[\text{Coll}(h, S)] = \sum_{\{x, y\} \subset S} \Pr[h(x) = h(y)] = \binom{|S|}{2} \frac{1}{n} \leq \frac{|S|^2}{2n} \leq \frac{n}{2}.$$

By Markov’s inequality, we have

$$\Pr_{h \in \mathcal{H}}[\text{Coll}(h, S) \geq n] \leq 1/2.$$

So, after on average two iterations of randomly choosing $h \in \mathcal{H}$, we find such a function $h : U \rightarrow [n]$ such that $\text{Coll}(h, S) \leq n$. We fix it from now on. Note that it is represented using only $O(\log n)$ bits.
Step 2. Next, for any $i \in [n]$ let $S_i = \{ x \in S : h(x) = i \}$. Observe that $\sum S_i = n$ and

$$\sum_{i=1}^{n} \left( \frac{|S_i|}{2} \right) = \text{Coll}(h, S) \leq n.$$ 

Let $n_i = |S_i|^2$. Note that $\sum n_i = 2\text{Coll}(h, S) + \sum |S_i| \leq 3n$. We will find hash functions $h_i : U \rightarrow [n_i]$ which are collision free on $S_i$. Choosing a uniform hash function from a pairwise independent set of hash functions $\mathcal{H}_i = \{ h : U \rightarrow [n_i] \}$ succeeds on average after two samples. So, we only need $O(n)$ time to find these functions. As each $h_i$ requires $O(\log n)$ bits to be represented, we need in total $O(n \log n)$ to represent all of them.

Let $A$ be an array of size $3n$. Let $\text{offset}_i = \sum_{j<i} n_j$. The sub-array $A[\text{offset}_i : \text{offset}_i + n_i]$ will be used to store the elements of $S_i$. Initially $A$ is empty. We set

$$A[\text{offset}_i + h_i(x)] = x \quad \forall x \in S_i.$$ 

Note that there are no collisions in $A$, as we are guaranteed that $h_i$ are collision free on $S_i$. We will also keep all of $\{ \text{offset}_i : i \in [n] \}$ in a separate array.

**Query.** To check whether $x \in S$, we do the following:

- Compute $i = h(x)$.
- Read $\text{offset}_i$.
- Check if $A[\text{offset}_i + h_i(x)] = x$ or not.

This can be computed using $O(\log n)$ bit operations.

**Space requirements.** The hash functions $h$ requires $O(\log n)$ bits. The hash functions $\{ h_i : i \in [n] \}$ require $O(n \log n)$ bits. The array $A$ requires $O(n \log n)$ bits.

**Setup time.** The setup algorithm is randomized, as it needs to find good hash functions. It has expected running time is $O(n \log n)$ bit operations.

- To find $h$ takes $O(n \log n)$ time, as this is how long it takes to verify that it is collision free.
- To find each $h_i$ takes $O(|S_i| \log n)$ time, and in total it is $O(n \log n)$ time.
- To set up the arrays of $\{ \text{offset}_i : i \in [n] \}$ and $A$ takes $O(n \log n)$ time.

**RAM model vs bit model.** Up until now, we counted bit operations. However, computers can operate on words efficiently. A model for that is the RAM model, where we can perform basic operations on $\log n$-bit words. In this model, it can be verified that the query time is $O(1)$ word operations, space requirements are $O(n)$ words and setup time is $O(n)$ word operations.
5 Bloom filters

Bloom filters allow for even more efficient data structures for set membership, if some errors are allowed. Let \( U \) be a universe, \( S \subset U \) a subset of size \(|S| = n\). Let \( h : U \to [m] \) be a uniform hash function, for \( m \) to be determined later. The data structure maintains a bit array \( A \) of length \( m \), initially set to zero. Then, for every \( x \in S \), we set \( A[h(x)] = 1 \). To check if \( x \in S \) we answer “yes” if \( A[h(x)] = 1 \). This has the following guarantees:

- No false negative: if \( x \in S \) we will always say “yes”.
- Few false positives: if \( x \notin S \), we will say “yes” with probability \( \frac{|\{i : A[i] = 1\}|}{m} \), assuming \( h \) is a uniform hash function.

So, if we set for example \( m = 2n \), then the probability for \( x \notin S \) that we say ”no” it at least \( 1/2 \). It is in fact more, since when hashing \( n \) elements to \( 2n \) values there will be collisions, so the number of 1’s in the array will be less than \( n \). In fact, to get probability \( 1/2 \) we only need \( m \approx 1.44n \). This is since the probability that \( A[i] = 1 \), over the choice of \( h \), is

\[
\Pr_h[A[i] = 0] = \Pr[h(x) \neq i, \forall x \in S] = \left(1 - \frac{1}{m}\right)^n \approx \exp(-n/m).
\]

So, for \( m = n/\ln(2) \approx 1.44n \), the expected number of 0’s in \( A \) is \( m/2 \).

Note that a bloom filter uses \( O(n) \) bits, which is much less than the \( O(n \log |U|) \) bits we needed for no errors. It can be shown that we don’t need \( h \) to be uniform, a \( O(\log n) \)-wise independent hash function gives the same guarantees, and it can be stored very efficiently (concretely, using only \( O(\log^2 n) \) bits).