Hashing II: Fingerprinting[1]

- **Back to Alice and Bob from Lecture 1.** Recall the problem we started the course with. Alice and Bob have bit-strings $\mathbf{x} \in \{0,1\}^n$ and $\mathbf{y} \in \{0,1\}^n$, and they wish to figure out if $\mathbf{x} = \mathbf{y}$? We saw that if they had access to *public randomness*, that is, a series of random bits on which both of them can agree upon, then Alice can send a dot-product of $\mathbf{x}$ with a random $\mathbf{r}$ which Bob checks with $\mathbf{y} \cdot \mathbf{r}$. If $\mathbf{x} \neq \mathbf{y}$, then the probability the dot-products are equal is $\frac{1}{2}$. Thus, to solve the problem with probability $1 - \delta$, Alice needed to send only $\lceil \ln(1/\delta) \rceil$ bits.

  We now show how Alice can solve the problem without assuming any public randomness on which Alice and Bob need agree. Instead she will use "private randomness". Using this, Alice will generate a small *sketch* or a *fingerprint* of her string $\mathbf{x}$. This information will allow Bob to form his sketch of $\mathbf{y}$, and accept if the sketches match. The Lecture 1 idea was similar, but the "sketch" needed the random vector $\mathbf{r}$ which couldn't be sent as that was n bits long.

- The idea is very similar to hashing. Alice wants to pick a function h from a family H which satisfies the following properties: (a) $h : \{0,1\}^n \to \{0,1,\ldots,\ell-1\}$ where $\ell \ll n$, (b) h is "small", that is, its description size is $\ll n$, and (c) if $\mathbf{x} \neq \mathbf{y}$, then $h(\mathbf{x}) \neq h(\mathbf{y})$ with high probability. This is precisely what a universal hash family H satisfies. ***Except***, the universe size here $N = 2^n$, and the size of the Carter-Wegman function, for instance, is $O(\lg N)$ which takes us back to the n regime. In other words even the UHF size $|H|$ is too large for this purpose. Something even smaller is needed.

- This construction uses a bit of number theory, in particular, a fact about prime numbers. Let $\pi(n)$ be the function denoting the *number* of prime numbers in the integers $\{1,2,\ldots,n\}$. We understand what $\pi(n)$ looks like very well. It is known that $\pi(n) \sim \frac{n}{\ln n}$, and is called the *prime number theorem*. The part that we need is $\pi(n) \geq \frac{n}{\ln n}$, or rather the following corollary

  For any positive integer k, the kth prime P_k is at most $2k \ln k$.

- Now we are ready to describe the "smaller" hash family. Let p be a prime. Define the function $h_p : \{0,1\}^n \to \{0,1,\ldots,p-1\}$ as

  $$h_p(\mathbf{x}) = \mathsf{int}(\mathbf{x}) \bmod p \quad \text{where } \mathsf{int}(x) \text{ is the integer whose binary representation is } \mathbf{x}$$

  Note that the size of the function and its output are both $O(\log p)$ bits. Now, let P_{2n} be the $(2n)$th prime. By the prime number theorem, $P_{2n} = O(n \ln n)$. Define $H := \{h_p : 1 \leq p \leq P_n,\ p \text{ prime}\}$. This is the family of "hash functions", or as it more commonly called, *fingerprint* functions, from which Alice chooses her hash. Note that $|H| = 2n$, by definition. And every function $h_p \in H$ takes $O(\ln n)$ bits to describe. Here is the main fact that resolves collisions.

  Lemma 1. Given $\mathbf{x} \neq \mathbf{y}$, $\mathbf{Pr}_{h_p \in H}\ [h_p(\mathbf{x}) = h_p(\mathbf{y})] \leq \frac{1}{2}$.

[1] *Lecture notes by Deeparnab Chakrabarty. Last modified : 11th April, 2021*
These have not gone through scrutiny and may contain errors. If you find any, or have any other comments, please email me at deeparnab@dartmouth.edu. Highly appreciated!

Remark: *Note that it is much weaker than what UHFs require. The range of the function has p elements, but the probability of collision is only bounded by $\frac{1}{2}$. But this suffices for the communication purpose.*

Before we prove the lemma, let's show the protocol between Alice and Bob.

procedure ALICE-EQ($\mathbf{x}$):
  for $i = 1$ to k **do**:
    Choose h_{p_i} randomly from H, that is, p_i a random prime between 1 and $2n$.
    Compute $a_i := h_{p_i}(\mathbf{x}) = \mathsf{int}(\mathbf{x}) \bmod p_i$. ▷ *$a_i$ is $O(\log n)$ bits long.*
  Send $\mathbf{a} := ((a_i, p_i)\ i = 1, \ldots, k)$ to Bob. ▷ *This is $O(k \log n)$ bits of communication.*

procedure BON-EQ($\mathbf{x}$): ▷ *Receives* $\mathbf{a}$
  Using $p_1, \ldots, p_k$, Bob computes $b_i := h_{p_i}(\mathbf{y}) = \mathsf{int}(\mathbf{y}) \bmod p_i$.
  Bob rejects if any $b_i \neq a_i$, and accepts if all are equal.

If $\mathbf{x} = \mathbf{y}$, we must have $a_i = b_i$ for all i. If $\mathbf{x} \neq \mathbf{y}$, then the probability all k of them are equal, by Lemma 1 is $\leq \frac{1}{2^k}$. Therefore,

Theorem 1. For any $\delta > 0$, Alice and Bob can solve the equality problem with error probability $\leq \delta$ communicating $k := O(\log n \log(1/\delta))$ bits, with private randomness.

- *Proof of Lemma 1.* $h_p(\mathbf{x}) = h_p(\mathbf{y})$ if $\mathsf{int}(\mathbf{x}) \equiv_p \mathsf{int}(\mathbf{y})$, or in other words, p divides $a := |\mathsf{int}(\mathbf{x}) - \mathsf{int}(\mathbf{y})|$. Since $\mathbf{x} \neq \mathbf{y}$, we get that a is a positive integer. Since both $\mathbf{x}$ and $\mathbf{y}$ are n-bit vectors, we get $a < 2^n$. Let $p_1, \ldots, p_t$ be the distinct primes between 1 and $2n$ which divide a. Thus, $a \geq \prod_{i=1}^{t} p_i \geq 2^t$ since each prime is ≥ 2. Indeed, this implies that $t < n$. Since p is chosen among the first $2n$ primes, the probability p is one of the p_i's is $\frac{t}{2n} < \frac{1}{2}$. □

- **Karp-Rabin Pattern Matching Algorithm.** In the pattern matching problem, we are given a string $\mathbf{x} \in \{0,1\}^n$ and a pattern string $\mathbf{y} \in \{0,1\}^m$, where you should think $n \gg m$. The objective is to figure out if $\mathbf{y}$ exists as a substring of $\mathbf{x}$. That is, does there exists $j \in [n]$ such that $\mathbf{x}_j = \mathbf{y}_1$, $\mathbf{x}_{j+1} = \mathbf{y}_2$, and so on, till $\mathbf{x}_{j+m+1} = \mathbf{y}_m$. Naively, this algorithm takes $O(nm)$ time. We now show a simple randomized $O(n+m)$ algorithm using fingerprinting.

  The idea is similar to the one given for equality. The algorithm selects a random prime p among the first k primes, where we set $k = n^2 m$ later. Next it computes $h_p(\mathbf{y})$. Then it does a single loop over $\mathbf{x}$, and in iteration j, it checks if $h_p(\mathbf{x}^{(j)}) =? h_p(\mathbf{y})$, where $\mathbf{x}^{(j)} = (\mathbf{x}_j, \mathbf{x}_{j+1}, \ldots, \mathbf{x}_{j+m+1})$.

  Note that if $\mathbf{x}^{(j)} = \mathbf{y}$, then the two fingerprints would be the same. If $\mathbf{x}^{(j)} \neq \mathbf{y}$, then the probability their fingerprints are the same is, as above, is $\leq \frac{m}{n^2 m} = \frac{1}{n^2}$. Therefore, by an union bound, the probability any of these $(n-m)$ fingerprints match is at most $\frac{1}{n}$.

  What is the running time of the algorithm? First note that the size of p is $O(\lg(n^2 m \lg(n^2 m))$ bit, which is $O(\log n)$. It takes around that time to evaluate $O(h_p(\mathbf{y}))$. How about computing $h_p(\mathbf{x}^{(j)})$. Next we use our final observation: we see that,

$$\mathsf{int}(\mathbf{x}^{(j+1)}) = 2 \cdot \left(\mathsf{int}(\mathbf{x}^{(j)}) - 2^{m-1}\mathbf{x}_j\right) + x_{j+m}$$

and therefore,

$$h_p(\mathbf{x}^{(j+1)}) = 2 \cdot \left(h_p(\mathbf{x}^{(j)} - 2^{m-1}\mathbf{x}_j \right) + x_{j+m} \bmod p$$

which can be calculated in $O(1)$ time (assuming p fits in a single word).