

1. **Hashing Large Universes, 35 points.** One awkward feature of k -wise independent hash families when applied to very large universes (for example if the universe consists of strings with at most a million characters, encoding decent sized text documents) is that with standard constructions, a member of a k -wise independent family would require $k \log_2(N)$ bits to store and would take $O(k \log(N))$ time to compute, where N is the universe size.

One solution to this is to first apply a relatively weak hash function $H : U \rightarrow [M]$ where $M = O(n^3)$ so that:

1. For any $S \subset U$ of size n , with reasonably high probability H causes no collisions on S .
 2. A member of the family of H can be stored in $O(\log \log(N) + \log(n))$ bits and can be computed in $O(\log(N))$ time.
- (a) Give an example of such a construction. Hint: It might be useful to operate in two stages to first map to a set of size $O(\log(N))$ and then to one of size n^3 . [25 points]

Solution: We operate in two stages. First, we interpret the universe as integers in the set $[N]$ and hash S modulo a random prime p chosen from the first $\lceil n^3 \log(N) \rceil$ many primes. Note that p (and thus the resulting new universe U') has size at most $O(n^3 \log(N) \log(n \log N))$ by the prime number theorem.

A collision between $x, y \in S$ with $x > y$ only occurs if $p|(x-y)$, which happens with probability at most $1/n^2$, since $x - y \leq N$ and thus can have at most $\log N$ prime divisors. A simple union bound over the $\binom{n}{2}$ pairs of elements in S upper bounds the probability of a collision in this first stage by $O(1/n)$.

For the second stage, we hash these new values down to a set of size $M = O(n^3)$ with a pairwise independent hash family. The probability of a single pair colliding in this step is at most $\binom{n}{2} \cdot \frac{1}{M} = O(1/n)$. Hence, the probability of a collision in either stage is at most $O(1/n)$.

We now analyze the storage and runtime. We can store the random prime p using $O(\log p) = O(\log \log(N) + \log(n))$ many bits, and the “standard constructions” of members of the two-wise independent hash families from the question’s description over our smaller universe U' can be stored in (asymptotically) the same amount of space. Since the elements of S take $O(\log N)$ many bits to store, we can take them modulo p in a comparable amount of time. Additionally, evaluating the pairwise independent hash function takes time $O(\log \log(N) + \log(n))$, so the total runtime is $O(\log N)$.

- (b) Show that this construction is optimal in the sense that for $N > m > n > 1$ that no hash family $H : [N] \rightarrow [m]$ consisting of fewer than $\log(N)/\log(m)$ functions can have the property that for any set $S \subset [N]$ of size n the probability that H causes a collision among the elements of S is less than $1/2$. [10 points]

Solution: Suppose by contradiction such a hash family $\mathcal{H} = \{h_1, h_2, \dots\}$ of size $T < \log_m(N)$ exists. For each $x \in [N]$, define the vector $v_x = (h_1(x), h_2(x), \dots, h_T(x))$. There are $m^T < N$ unique vectors, so the pigeon-hole principle guarantees two distinct inputs $x, y \in [N]$ such that every hash function $h \in \mathcal{H}$ satisfies $h(x) = h(y)$. Thus, every set S containing both x and y causes a collision, a contradiction.

2. **Cuckoo Hashing with 3-wise independence, 35 points.** Consider a universe $U = \mathbb{F}_2^T$ and identify $[m]$ with \mathbb{F}_2^t . Consider a family of hash functions h by letting h be a uniform random affine linear transformation from $\mathbb{F}_2^T \rightarrow \mathbb{F}_2^t$. In particular, $h(x) = Ax + b$ where A is a $t \times T$ matrix over \mathbb{F}_2 , and b is an \mathbb{F}_2^t vector.

- (a) Show that this defines a 3-wise independent hash family. [10 points]

Solution: Let $x_1, x_2, x_3 \in \mathbb{F}_2^T$ be three distinct vectors, and let $y_1, y_2, y_3 \in \mathbb{F}_2^t$ be three (possibly non-distinct) vectors. We want to show that

$$\Pr_h[h(x_1) = y_1 \wedge h(x_2) = y_2 \wedge h(x_3) = y_3] = \frac{1}{m^3}.$$

Note that it suffices to prove this coordinate-wise. To that end, look at the first coordinate, where we write $h(x)^{(1)} = \langle a, x \rangle + b$ with $a, b \in \mathbb{F}_2^T$. Consider

$$\begin{aligned} \Pr_h[h(x_1)^{(1)} = y_1^{(1)} \wedge h(x_2)^{(1)} = y_2^{(1)} \wedge h(x_3)^{(1)} = y_3^{(1)}] \\ = \Pr_h[\langle a, x_1 \rangle + b = y_1^{(1)} \wedge \langle a, x_2 \rangle + b = y_2^{(1)} \wedge \langle a, x_3 \rangle + b = y_3^{(1)}]. \end{aligned}$$

We observe that this corresponds to a system of three linearly independent equations: there cannot be a dependence involving all three equations, since $3b \neq 0$ whenever $b = 1$, nor can there be a dependence involving exactly two equations, since that would force (say) $x_1 = x_2$, violating our distinctness assumption. Thus, the above probability equals $(1/2)^3$, which implies the result.

- (b) Show that if this family is used to implement Cuckoo Hashing for a carefully chosen set $S \subset U$ with $|S| = O(m^{5/6})$ that after picking a random hash function h from this family, one will be forced to rehash with at least constant probability. Hint: Let S be a random subset of a subspace of dimension $t + 2$. [25 points]

Solution: Recall that one is forced to rehash if the graph corresponding to the two hash functions has a connected component with at least two cycles (in which case the number of edges exceeds the number of vertices). For this problem, we will choose an S such that this graph has a copy of the complete bipartite graph $K_{2,3}$.

Let $V = \mathbb{F}_2^{t+2}$, and consider a random subset $S \subseteq V$ where each element of V is included in S independently with probability $n/|V| = n/(4m)$ for some

parameter n to be set later. Observe that $\mathbb{E}|S| = n$, so $|S| = O(n)$ with some large positive probability by Markov's inequality.

Let $f(x) = Ax + c$ and $g(x) = Bx + d$ be two uniformly random affine linear transformations from \mathbb{F}_2^{t+2} to \mathbb{F}_2^t . By the rank-nullity theorem, the kernels of both A and B have dimension at least two.

If $\ker(A) \cap \ker(B) = \{0\}$, then we can select two linearly independent vectors $x, y \in \ker(A)$ and some nonzero vector $z \in \ker(B)$. Note that $Az, Bx, By \neq 0$ and $Bx \neq By$. It remains to find a vector w so that $w, w+x, w+y, w+z, w+z+x, w+z+y$ are all in S , as this corresponds to the aforementioned $K_{2,3}$ in the cuckoo graph (since adding x or y does not affect f 's hash, and adding z does not affect g 's hash).

Otherwise, $\ker(A), \ker(B)$ intersect at some nonzero vector x , and we can choose $y \in \ker(A)$ to be linearly independent from x , and $z = x$. In this case, the configuration above reduces to the simpler four point configuration $w, w+x, w+y, w+x+y$.

Henceforth, we will focus on finding patterns of the form $w, w+x, w+y, w+z, w+z+x, w+z+y$; having fewer points (as in the 4 point configuration) will only make this easier. Each one of these 6-tuples conflicts with at most 8 others (corresponding to the $\text{span}(x, y, z)$), so we can greedily find $|V|/8 = m/2$ independent translates of the desired form, each of which is included in S with probability $(n/(4m))^6$. Thus, we can find such a w with probability at least

$$1 - \left(1 - \left(\frac{n}{4m}\right)^6\right)^{m/2} \geq 1 - \exp\left(-\left(\frac{n}{4m}\right)^6 \cdot \frac{m}{2}\right),$$

which can be made arbitrarily close to one by choosing $n \geq Cm^{5/6}$ with a sufficiently large constant. By a union bound, we have with positive probability that there exists some S of size $O(m^{5/6})$ that forces a rehashing.

Note that if n is bigger by an additional logarithmic factor, the probability of failure is $1/\text{poly}(n)$, and we can union bound over all triples of x, y, z to show that there is some construction so that we will *always* need to rehash.

3. **Fingerprinting and Communication Complexity, 30 points.** In communication complexity, two players (traditionally named Alice and Bob) are each given half of the input to some function $F(X, Y)$. They take turns sending one bit of information to the other, and eventually want one to declare the correct value of the function F . The goal is to minimize the number of bits sent regardless of the amount of computation required.

In particular, we will look at the equality problem. Alice is given an n bit string X and Bob is given an n bit string Y and they want to compute the function $F(X, Y)$, which is 1 if $X = Y$ and 0 otherwise.

- (a) Prove that for deterministic protocols, n bits of communication are required. Hint: Consider the full communication transcripts if $X = Y = s$ for various values of s . [15 points]

Solution: Suppose by contradiction there exists a deterministic protocol which sends only $n - 1$ bits of communication. That is, there are at most 2^{n-1} different transcripts the protocol can produce. Consider the 2^n transcripts produced by inputs of the form (X, X) for all possible $X \subseteq \{0, 1\}^n$. By the pigeonhole principle, there must exist two distinct X_1, X_2 such that the protocol produces the same transcript on (X_1, X_1) and (X_2, X_2) .

The key observation is that this implies the protocol also produces the same transcript on (X_1, X_2) , since from (say) Alice's perspective, the messages she is receiving are indistinguishable from those she receives when the input is (X_1, X_1) , so she sends messages as though that were in fact the case. (A similar argument applies to Bob.) Notice, however, that $F(X_1, X_1) \neq F(X_1, X_2)$, so they cannot produce the same transcripts if the protocol is correct. This is a contradiction.

- (b) Show that only 1 bit of communication is required if Alice and Bob have a source of shared randomness (i.e. there is a random string that Alice and Bob can both read without this counting as communication). In particular, you should design what one might call a coRP algorithm where if $X = Y$, they definitely return 1, but if $X \neq Y$ there is at least a 50% chance that they return 0. [15 points]

Solution: The players can interpret the shared random string as encoding a hash function $h: \{0, 1\}^n \rightarrow \{0, 1\}$ by viewing the random bits as the outputs of inputs in some predefined ordering. Alice sends the encoding of her input $h(X)$ to Bob, and he checks whether his input produces the same hash (i.e., whether $h(X) = h(Y)$). If it does he returns 1, and otherwise he returns 0.

If $X = Y$, the two hashes are always equal, so the players will always return 1. If $X \neq Y$, the two hashes will differ with probability $1/2$, so they will return 0 with 50% probability.

(A more randomness-efficient approach is to view the bits as a vector $Z \in \{0, 1\}^n$, and then have Alice send the inner product $\langle X, Z \rangle$.)