NOTATION: for a given positive integer $n$, let $[n] := \{0, \ldots, n-1\}$.

1. **Composing universal hash functions.** Let $\{h_k\}_{k \in K}$ be an $\epsilon$-universal family of hash functions from $U$ to $V$ (that is, $h_k : U \rightarrow V$ for each $k \in K$). Let $\{\phi_{\lambda}\}_{\lambda \in \Lambda}$ be a $\delta$-universal family of hash functions from $V$ to $W$. Show that $\{\phi_{\lambda} \circ h_k\}_{(k, \lambda) \in K \times \Lambda}$ is an $(\epsilon + \delta)$-universal family of hash functions from $U$ to $W$.

Here, the function $\phi_{\lambda} \circ h_k$ maps $a \in U$ to $\phi_{\lambda}(h_k(a)) \in W$.

2. **Reducing the key size for universal hash functions.** Let $m$ and $t$ be positive integers, and let $0 < \delta < 1$. Show how to construct an $\epsilon$-universal family $\{h_k\}_{k \in K}$ of hash functions from $[m]^t$ to $[m]$, such that

$$\epsilon \leq \frac{1}{m}(1 + \delta) \quad \text{and} \quad \log |K| = O(\log m + \log t + \log(1/\delta)).$$

Hint: use the result of the previous exercise, as well as the polynomial evaluation hash discussed in class, and the hash function described in Theorem 11.5 in the textbook (which we also mentioned in class).

Note: compare this to the constructions we have seen for universal families (i.e., $\delta = 0$), where $\log |K| \approx t \log m$.

3. **A pairwise independent family.** Let $m$ be a prime, $U := \mathbb{Z}_m$, and $K := \mathbb{Z}_m^{t+1}$.

For $a = (a_1, \ldots, a_t) \in U$ and $k = (k_0, k_1, \ldots, k_t)$, define

$$h_k(a) := k_0 + \sum_{i=1}^t a_i k_i \in \mathbb{Z}_m.$$ 

Show that $\{h_k\}_{k \in K}$ is a pairwise independent family of hash functions from $U$ to $\mathbb{Z}_m$.

4. **Another pairwise independent family.** Let $m$ be a power of 2, $t$ be a positive integer, $U := [m]^t$, and $K := [m^2]^{t+1}$. For $a = (a_1, \ldots, a_t) \in U$ and $k = (k_0, k_1, \ldots, k_t) \in K$, let

$$h_k(a) := \left\lfloor \left( k_0 + \sum_{i=1}^t a_i k_i \right) \mod m^2 \right\rfloor \div m.$$ 

Show that $\{h_k\}_{k \in K}$ is a pairwise independent family of hash functions from $U$ to $[m]$.

Hint: the following fact may be useful. Let $a, b, m$ be integers with $m > 0$, and let $d := \gcd(a, m)$.

The congruence $ax \equiv b \pmod{m}$ has a solution $x$ if and only if $d \mid b$, in which case it has exactly $d$ distinct solutions modulo $m$.

Note that on a 64-bit machine, if $m = 2^{32}$, then it is very easy to evaluate these hash functions, using just multiplications, additions, shifts, and masks.

5. **Longest palindromic substring.** You are to design an algorithm for the following problem.

The input is an array $A[1..n]$ of 8-bit ASCII characters. A palindromic substring of $A$ is a substring $A[i..j]$ of $A$ that is a palindrome, i.e., it reads the same forwards or backwards.
course, any substring of length 1 is a palindromic substring. Your algorithm is to find a longest palindromic substring.

The output of your algorithm is just the indices $i$ and $j$ that define the substring $A[i..j]$. Your algorithm may be probabilistic, and should run in expected time $O(n \log n)$. Your algorithm should also use just constant space — not including the input array $A$ (which is read only).

Your algorithm should use universal hashing. Here are the rules of the game, in terms of counting time and space. Calculations on array indices and reading an element of $A$ cost 1 time unit.

Storage for an array index costs 1 space unit. To implement a hash function, your algorithm may choose a prime $p$ of any bit length, as long as that bit length is $O(\log n)$. You are charged 1 time unit for selecting $p$. Given such a prime $p$, each of the following operations cost 1 time unit:

- generating a random element in $\mathbb{Z}_p$
- addition, subtraction, multiplication, or division in $\mathbb{Z}_p$

Also, you are charged 1 space unit for each element of $\mathbb{Z}_p$ that your algorithm stores in memory.

Hints:

- Use a variation of the Karp/Rabin pattern matching algorithm.
- Try solving the following problem first: given $A[1..n]$ as above and $\ell \in \{1, \ldots, n\}$, determine if $A$ has a palindromic substring of length $\ell$ in expected time $O(n)$ and constant space.

Note: there is actually a deterministic $O(n)$ time algorithm for this, but it uses additional space $O(n)$.

6. Hash 'til you crash. Assume $|\mathcal{U}| > m$, and consider fixed, distinct data items $a_1, \ldots, a_{m+1} \in \mathcal{U}$. Suppose a hash function $h : \mathcal{U} \to [m]$ is chosen at random from some family of hash functions. Let $X$ be the least positive integer $i$ such that $h$ maps two items among $a_1, \ldots, a_i$ to the same slot; that is, if we insert $a_1, \ldots, a_{m+1}$ one at a time into an initially empty hash table, then $X$ represents the number of insertions we perform until some slot contains 2 items.

Under the uniform hashing assumption, show that $E[X] = \Theta(m^{1/2})$. Hint: use the identity $E[X] = \sum_{j \geq 1} \Pr[X \geq j]$.

7. Cuckoo hashing. Give detailed proofs of the following assertions. The setting is as follows. We have a cuckoo graph $G$ with $m$ vertices $0, \ldots, m-1$ (corresponding to hash table slots) and $n$ edges $e_1, \ldots, e_n$ (corresponding to data items). Each (undirected) edge $e_i$ is chosen at random as $\{u_i, v_i\}$ by selecting the endpoints $u_i$ and $v_i$ uniformly and independently from $\{0, \ldots, m-1\}$.

Define $\alpha := n/m$, which is the “load factor”.

(a) For every $k \geq 1$, the probability $p_k$ that $G$ contains a simple cycle of length $k$ is at most $(2\alpha)^k/k$.

Here, a simple cycle of length $k$ is a path $(s_0, s_1, \ldots, s_{k-1}, s_0)$, where the vertices $s_0, \ldots, s_{k-1}$ are distinct.

(b) With $p_k$ defined as in part (a): $p_1 \leq \alpha$ and $p_2 \leq \alpha^2$.

Note that these bounds are a bit better than the general bounds obtained in part (a) for $k = 1$ and $k = 2$.

(c) For every fixed vertex $s_0$, and every $\ell \geq 1$, the probability that $G$ contains a simple loop of length $\ell$ starting at $s_0$ is at most $\ell(2\alpha)\ell/m$.

Here, a simple loop of length $\ell$ starting at $s_0$ is a path $(s_0, \ldots, s_{\ell-1}, s_j)$, where $j < \ell$ and the vertices $s_0, \ldots, s_{\ell-1}$ are distinct.