Basic Algorithms — Fall 2020 — Problem Set 8
Due: Wed, Dec 9, 11am

1. **Half-priced hash.** In class, we studied a family of hash functions based on taking inner products. That family was a family of hash functions from keys $\mathbb{Z}_m^t$ to slots $\mathbb{Z}_m$ indexed by seeds $\mathbb{Z}_m^t$, where m is prime. For each seed $\lambda = (\lambda_1, \ldots, \lambda_t) \in \mathbb{Z}_m^t$, and each key $a = (a_1, \ldots, a_t) \in \mathbb{Z}_m^t$, the hash function was defined as $h_\lambda(a) := \sum_i^t a_i \lambda_i$, which requires t multiplications to evaluate. In this problem, you are to analyze a variant hash which cuts the number of multiplications in half. In most computing environments, multiplications are much more expensive than addition, so this can speed up the hash function evaluation significantly.

   Assume t is even, so $t = 2s$. Hash function seeds and keys have the same structure as above, but the hash function is defined as follows:

$$h'_\lambda(a) := \sum_{i=1}^{s} (a_{2i-1} + \lambda_{2i-1})(a_{2i} + \lambda_{2i}).$$

   So, for example, for $t = 4$, we have

$$h'_\lambda(a) = (a_1 + \lambda_1)(a_2 + \lambda_2) + (a_3 + \lambda_3)(a_4 + \lambda_4).$$

   Your task is to show that the family of hash functions $\{h'_\lambda\}_{\lambda \in \mathbb{Z}_m^t}$ is a universal family.

   Hint: Your proof should mimic the one given in class for the inner-product based family. Namely, consider two distinct keys $a = (a_1, \ldots, a_t)$ and $b = (b_1, \ldots, b_t)$, and show that the number of seeds $(\lambda_1, \ldots, \lambda_t)$ which satisfy

$$\sum_{i=1}^{s} (a_{2i-1} + \lambda_{2i-1})(a_{2i} + \lambda_{2i}) = \sum_{i=1}^{s} (b_{2i-1} + \lambda_{2i-1})(b_{2i} + \lambda_{2i})$$

   is at most m^{t-1}. To keep the notation simple, you may first want to do the calculation for the case $t = 4$.

2. **Mod-free hash.** In class, most of the hash functions we looked at required arithmetic mod m, where m was a prime. This exercise looks a family of hash functions where this is not necessary, which can result in a significantly more efficient implementation.

   Let k, ℓ, and t be positive integers. Let $\mathcal{U} := [0\,..\,2^k)^t$, $\mathcal{V} := [0\,..\,2^\ell)$, and $\Lambda := [0\,..\,2^{k+\ell})^t$. We define a family of hash functions $\{h_\lambda\}_{\lambda \in \Lambda}$ from $\mathcal{U}$ to $\mathcal{V}$ as follows. For $a = (a_1, \ldots, a_t) \in \mathcal{U}$ and $\lambda = (\lambda_1, \ldots, \lambda_t) \in \Lambda$, we define

$$h_\lambda(a) := \left\lfloor \left((a_1\lambda_1 + \cdots + a_t\lambda_t) \bmod 2^{k+\ell} \right) / 2^k \right\rfloor.$$

   That is, if we write $\sum_i a_i\lambda_i = L2^{k+\ell} + M2^k + R$, where $0 \le R < 2^k$ and $0 \le M < 2^\ell$, then $h_\lambda(a) = M$. In other words, $h_\lambda(a)$ consists of the "middle bits", i.e., bits k through $k + \ell - 1$, of the inner product $\sum_i a_i\lambda_i$.

   To see why this can be efficiently implemented, suppose that $k = 10$ and $\ell = 20$. Then using 32-bit unsigned arithmetic, we can compute $sum \leftarrow \sum_i a_i\lambda_i$ using t integer multiplications and additions, and we can then compute the hash as $hash \leftarrow (sum \;\&\; (2^{30} - 1)) \gg 10$. Make sure you understand why this works before proceeding.

   Your goal is to show that that $\{h_\lambda\}_{\lambda \in \Lambda}$ is a $2^{-\ell+1}$-universal family of hash functions.

   Here is an outline you should follow:

   (a) Suppose $h_\lambda(a) = h_\lambda(b)$, where $a = (a_1, \ldots, a_t)$, $b = (b_1, \ldots, b_t)$, and $\lambda = (\lambda_1, \ldots, \lambda_t)$. Let $c_i := a_i - b_i$ for $i = 1, \ldots, t$. Show that for some integer d, with $|d| < 2^k$, we have

$$c_1\lambda_1 + \cdots + c_t\lambda_t \equiv d \pmod{2^{k+\ell}}. \qquad (*)$$

   (b) Further suppose that $a \neq b$, so that $c_{i'} \neq 0$ for some $i' \in [1\,..\,t]$. Let 2^j be the largest power of two that divides all of the c_i's. Show that $j < k$ and that $2^j \mid d$.

   Note: from (a) and (b), it follows that

$$d \in S := \{0, \pm 1 \cdot 2^j, \pm 2 \cdot 2^j, \ldots, \pm(2^{k-j} - 1) \cdot 2^j\}.$$

   In particular, $|S| = (2^{k-j} - 1) + (2^{k-j} - 1) + 1 = 2^{k-j+1} - 1$.

(c) Without loss of generality, we may assume that $c_1 = 2^j f$, where f is odd (we just need to re-arrange indices if necessary). Argue that for every choice of $d \in S$, and every choice of $\lambda_2, \ldots, \lambda_t \in [0\,..\,2^{k+\ell})$, there are exactly 2^j choices of $\lambda_1 \in [0\,..\,2^{k+\ell})$ that satisfy the congruence $(*)$.

Hint: Use Theorem 2.8 in the Number Theory Primer.

(d) Conclude that $\{h_\lambda\}_{\lambda \in \Lambda}$ is a $2^{-\ell+1}$-universal family of hash functions.

Hint: first, argue that for every possible d, there are $2^{(k+\ell)(t-1)+j}$ choices for the λ_i's that satsify $(*)$; from this, argue that the total number of choices for that λ_i's that can satisfy $(*)$ for *any* d is at most $2^{(k+\ell)(t-1)+k+1}$.

3. **Massive pile-up.** Suppose we hash n keys into a hash table using a randomly chosen seed for a universal family of hash functions. Let M be the maximum number of keys that get hashed to any one slot. In class, we called M the "max load", and we showed that $\mathrm{E}[M] = O(\sqrt{n})$, assuming that the load factor (# keys / # slots) was $O(1)$. The goal of this exercise is to show that this upper bound cannot be significantly improved.

Let $\mathcal{V} = \{1, \ldots, n\}$. We define a family of hash functions from $\mathcal{V}$ to $\mathcal{V}$ as follows. Let $k \in [1\,..\,n]$ be a parameter (determined below). For each subset $I \subseteq \mathcal{V}$ of carnality k, we define the hash function $h_I : \mathcal{V} \to \mathcal{V}$ by the following property: h_I maps each element of I onto 1, and h_I maps $\mathcal{V} \setminus I$ injectively (i.e., in a one-to-one fashion) into $\mathcal{V} \setminus \{1\}$. Note that h_I is not uniquely determined by this property, but we can always choose one h_I satisfying this property (you should verify this). Define

$$\mathcal{H} := \left\{h_I\right\}_{I \subseteq \mathcal{V}, |I| = k} .$$

Argue that $\mathcal{H}$ is universal if $k \leq \sqrt{n-1}$.

Hints: How many seeds I are there, as a function of n and k? For fixed $a, b \in \mathcal{V}$ with $a \neq b$, how many seeds I are there with $h_I(a) = h_I(b)$, again, as a function of n and k?

Discussion. Observe that if we hash all n elements in $\mathcal{V}$, then no matter which hash function h_I we choose, k elements get hashed to slot 1. Thus, for this family of universal hash functions with $k = \lfloor\sqrt{n-1}\rfloor$, we get a *lower bound* of $\Omega(\sqrt{n})$ on the expected max load, which matches the $O(\sqrt{n})$ upper bound obtained in class. Even though this family of hash functions is rather artificial, similar lower bounds hold for more natural universal families of hash functions. For example, for the "inner product hash" discussed in class, there is a lower bound of $\Omega(n^{1/3})$ on the expected max load [Alon, *et al*, "Is Linear Hashing Good?", STOC (1997)]. That paper also gives $\Omega(\sqrt{n})$ lower bounds for other natural families of universal hash functions.

4. **Common substring detection.** Consider the following problem. Given two strings a and b of length n and a parameter $t < n$, determine if a and b have a common substring of length t. A naive algorithm for this problem takes time $O(n^2 t)$. Here is a much faster randomized algorithm for this problem that is based on the Karp/Rabin algorithm. Let $\{h_\lambda\}_{\lambda \in \mathbb{Z}_m}$ be the polynomial hash family used in Karp/Rabin that hashes strings of length t to $\mathbb{Z}_m$, where m is a prime. The algorithm chooses a random seed $\lambda \in \mathbb{Z}_m$ and then runs as follows:

1. for i in $[1\,..\,n-t+1]$ do $r_i \leftarrow h_\lambda(a_i \cdots a_{i+t-1})$
2. for j in $[1\,..\,n-t+1]$ do $s_j \leftarrow h_\lambda(b_j \cdots b_{j+t-1})$
3. if $r_i = s_j$ for some $i, j \in [1\,..\,n-t+1]$
   then return *true*
   else return *false*

(a) Show that if a and b have a common substring of length t, then the algorithm always outputs *true*.

(b) Show that if a and b do *not* have a common substring of length t, then the algorithm outputs *true* with probability *at most* $(n-t+1)^2(t-1)/m$.

(c) We can implement Steps 1 and 2 so that they run in time $O(n)$ using the rolling hash technique, assuming m is small enough so that we can perform arithmetic operations in $\mathbb{Z}_m$ in constant time. For example, if we choose $m \approx 2^{64}$, the error probability in part (b) is reasonably small, but we can implement arithmetic in $\mathbb{Z}_m$ in constant time on a 64-bit machine.

A naive implementation of Step 3 would run in time $O(n^2)$. Your task is to show how to implement Step 3 probabilistically in *expected* time $O(n)$.

To this end, you should make use of the following fact.

Let m be a prime number as above, and let k be an arbitrary positive integer. Let

$$\Lambda := [1\,..\,m) \times [0\,..\,m).$$

Then the family of hash functions

$$\{g_{u,v}\}_{(u,v)\in\Lambda},$$

where each $g_{u,v} : [0\,..\,m) \to [0\,..\,k)$ is defined by

$$g_{u,v}(a) := \big(\ (ua+v) \bmod m\ \big) \bmod k,$$

is universal.

Show how to use such a family of hash functions to implement Step 3 in *expected* time $O(n)$. Among other things, you should specify how to choose k. You may use any facts regarding hash functions and hash tables proved in class.

5. **2D hash *[Food for thought (will not be graded)].*** In class, we presented a $(t-1)/m$-universal hash family based on polynomial evaluation. This exercise develops a two-dimensional variant. The universe of keys $\mathcal{U}$ consists of all $t \times t$ matrices over $\mathbb{Z}_m$, where m is prime. We write such a matrix $A \in \mathcal{U}$ as $A = (a_{ij})$, where the indices i and j run from 0 to $t-1$. The set of seeds Λ consists of pairs $(\lambda_1, \lambda_2) \in \mathbb{Z}_m \times \mathbb{Z}_m$. For $\lambda = (\lambda_1, \lambda_2) \in \Lambda$ and $A = (a_{ij}) \in \mathcal{U}$, define

$$h_\lambda(A) = \sum_{i=0}^{t-1}\sum_{j=0}^{t-1} a_{ij}\lambda_1^i\lambda_2^j \in \mathbb{Z}_m. \tag{1}$$

For example, if $t = 3$, we have

$$\begin{aligned} h_\lambda(A) = \quad & a_{00} + a_{01}\lambda_2 + a_{02}\lambda_2^2 \\ + \ & a_{10}\lambda_1 + a_{11}\lambda_1\lambda_2 + a_{12}\lambda_1\lambda_2^2 \\ + \ & a_{20}\lambda_1^2 + a_{21}\lambda_1^2\lambda_2 + a_{22}\lambda_1^2\lambda_2^2\ . \end{aligned}$$

Show that $\{h_\lambda\}_{\lambda\in\Lambda}$ is ϵ-universal for $\epsilon := 2(t-1)/m$.

Hint: suppose $h_\lambda(A) = h_\lambda(B)$, then $h_\lambda(C) = 0$, where $C := A - B$. If $C = (c_{ij})$, re-write $h_\lambda(C)$ as

$$h_\lambda(C) = \sum_{i=0}^{t-1} \lambda_1^i \Big(\sum_{j=0}^{t-1} c_{ij}\lambda_2^j \Big).$$

For $t = 3$, this looks like this:

$$\begin{aligned} h_\lambda(C) = \quad & (\ c_{00} + c_{01}\lambda_2 + c_{02}\lambda_2^2\ ) \\ + \ & \lambda_1(\ c_{10} + c_{11}\lambda_2 + c_{12}\lambda_2^2\ ) \\ + \ & \lambda_1^2(\ c_{20} + c_{21}\lambda_2 + c_{22}\lambda_2^2\ )\ . \end{aligned}$$

You want to show that if $A \neq B$, then the number of pairs $\lambda = (\lambda_1, \lambda_2)$ such that $h_\lambda(C) = 0$ is at most ϵm^2, where $\epsilon = 2(t-1)/m$.

To do this, you should make use of fact that any polynomial of degree at most $t-1$ over $\mathbb{Z}_m$ has at most $t-1$ roots. You will need to use this fact twice: once with

$$f(Y) = \sum_{j=0}^{t-1} c_{i_0 j} Y^j \quad \text{for some } i_0 \in [0\,..\,t),$$

and once with

$$g_{\lambda_2}(X) = \sum_{i=0}^{t-1} X^i \Big(\sum_{j=0}^{t-1} c_{ij}\lambda_2^j \Big) \quad \text{for some } \lambda_2 \in \mathbb{Z}_m.$$

6. **2D pattern matching** ***[Food for thought (will not be graded)].*** In the 2D pattern matching problem, you are given an $n \times n$ array A and a $t \times t$ array B, where $t \leq n$, and you want to determine if B appears as a subarray within A. For example, the array

$$B = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

appears as a subarray of

$$A = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & \mathbf{1} & \mathbf{2} & 3 \\ 2 & \mathbf{3} & \mathbf{4} & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}.$$

(a) Show how to compute the 2D hash h_{λ_1,λ_2} of the previous exercise on all of the $t \times t$ subarrays of A in time $O(n^2)$.

Hint: you will have to somehow adapt the "rolling hash" idea of Karp/Rabin to the 2D hash. Assume $A = (a_{ij})$, where the row indices i and column indices j both run from 1 to n. For $i = 1, \ldots, n$ and $j = 1, \ldots, n - t + 1$, let

$$\mu_{ij} := a_{ij} + \lambda_2 a_{i,j+1} + \cdots + \lambda_2^{t-1} a_{i,j+t-1}.$$

First, show how to compute all of the μ_{ij}'s in time $O(n^2)$ using the rolling hash technique. Second, using the μ_{ij}'s, use the rolling hash technique again to compute the 2D hash h_{λ_1,λ_2} on all of the $t \times t$ subarrays of A in time $O(n^2)$.

(b) Use your algorithm from part (a) to adapt the Karp/Rabin algorithm to solve the 2D pattern matching problem. The expected running time of your algorithm should be $O(n^2 + n^2 t^3/m)$, where m is the prime used in the 2D hash function. For reasonable choices of t and m, the first term will dominate, and so the expected running time will be $O(n^2)$.