

## CS 170 Homework 15

Due Monday 5/4/2026, at 10:00 pm (grace period until 11:59pm)

### Study Group

List the names and SIDs of the members in your study group. If you have no collaborators, explicitly write “none”.

### 1 Universal Hashing (Solo Question; 10 points)

Let  $\mathcal{H}$  be a family of hash functions in which each  $h \in \mathcal{H}$  maps the universe  $\mathcal{U}$  of keys to  $[m] := \{0, 1, \dots, m-1\}$ .  $\mathcal{H}$  is *universal* if for every  $x \neq y \in \mathcal{U}$ , the chance that  $h(x) = h(y)$  when we sample  $h$  uniformly at random from  $\mathcal{H}$  is at most  $1/m$ .

Consider the following family of hash functions from  $[m] \times [m]$  to  $[m]$ : Let  $h_{a,b}(x_1, x_2) = a \cdot x_1 + b \cdot x_2 \pmod m$ , and let  $\mathcal{H} = \{h_{a,b} | a, b \in [m]\}$ .

- If  $m$  is prime, show that this family is universal.
- If  $m$  is composite, show that this family is not universal.

#### Solution:

- For every  $(x_1, x_2) \neq (y_1, y_2)$ , so that either  $x_1 \neq y_1$  or  $x_2 \neq y_2$ , we want to show that

$$h_{a,b}(x_1, x_2) - h_{a,b}(y_1, y_2) = a \cdot (x_1 - y_1) + b \cdot (x_2 - y_2) \pmod m$$

equals 0 with probability at most  $1/m$ . Assume  $x_1 \neq y_1$  (the  $x_2 \neq y_2$  case is similar). Then the values  $a \cdot (x_1 - y_1)$  for  $a \in [m]$  are all distinct mod  $m$ , as the difference  $(a - a') \cdot (x_1 - y_1)$  of any two of these values is the product of two integers that are both  $< m$ , and hence is not divisible by the prime  $m$ . Thus regardless of the value of  $b \cdot (x_2 - y_2)$ , for a random choice of  $a$  then  $h_{a,b}(x_1, x_2) - h_{a,b}(y_1, y_2)$  will be a uniformly random number mod  $m$ , and in particular equals 0 with probability  $1/m$ .

- Let  $n < m$  be the greatest divisor of  $m$ , so that in particular  $n \geq \sqrt{m}$ . Then  $h_{a,b}(0, 0) = 0$  for every  $a, b$ , and  $h_{a,b}(n, 0) = 0$  whenever  $a$  is a multiple of  $m/n$ . As there are  $n$  multiples of  $m/n$  in  $[m]$ , the chance that both hashes equal 0 is  $n/m \geq 1/\sqrt{m} > 1/m$ , so the family is not universal.

## 2 Super-Universal Hashing (10 points)

Let  $\mathcal{H}$  be a class of hash functions in which each  $h \in \mathcal{H}$  maps the universe  $\mathcal{U}$  of keys to  $[m] := \{0, 1, \dots, m-1\}$ . Recall the definition of universal  $\mathcal{H}$  from Question 1.

We say that  $\mathcal{H}$  is super-universal if for every fixed  $x \neq y \in \mathcal{U}$ , the pair  $(h(x), h(y))$  is equally likely to be any of the  $m^2$  pairs of elements of  $[m]$  when  $h$  is sampled uniformly at random from  $\mathcal{H}$ . (The probability is taken only over the random choice of the hash function.)

- (a) Show that if  $\mathcal{H}$  is super-universal, then it is universal.
- (b) Suppose that you choose a hash function  $h \in \mathcal{H}$  uniformly at random. Your friend, who knows  $\mathcal{H}$  but does not know which hash function you picked, tells you a key  $x$ , and you tell her  $h(x)$ . Can your friend then find some  $y \neq x$  such that  $h(x) = h(y)$  with probability greater than  $1/m$  (over your choice of  $h$ ) if:
  - (i)  $\mathcal{H}$  is universal?
  - (ii)  $\mathcal{H}$  is super-universal?

In each case, either give a choice of  $\mathcal{H}$  which allows your friend to find a collision, or prove that they cannot for any choice of  $\mathcal{H}$ .

### Solution:

- (a) If  $\mathcal{H}$  is super-universal, then the pair  $(h(x), h(y))$  takes on all  $m^2$  possible values with equal probability, and in  $m$  of these values we have  $h(x) = h(y)$ . So  $h(x) = h(y)$  with probability  $m/m^2 = 1/m$ , as desired.
- (b)
  - (i) For a prime  $m$ , consider the hash functions  $h_a(x_1, x_2) = ax_1 + x_2 \pmod m$  for all  $a \in [m]$ . This family  $\{h_a\}$  is universal by similar reasoning as in Question 1(a). However, if you tell your friend  $h_a(1, 0) = a$ , then your friend can set  $y = (0, a)$ , to obtain the collision  $h_a(0, a) = a = h_a(1, 0)$ .
  - (ii) Since the pair  $(h(x), h(y))$  is equally likely to be any of the  $m^2$  possibilities, conditioned on the value of  $h(x)$  then  $h(y)$  is still a uniformly random element of  $[m]$  for every choice of  $y$ . Hence the chance that  $h(x) = h(y)$  is  $1/m$ .

### 3 Streaming Integers (10 points)

In this problem, we are given an infinite stream of positive integers  $x_1, x_2, \dots$ , and we have to perform some computation after each new integer is given. Since we may see many integers, we want to limit the amount of memory we have to use in total. For all of the parts below, give a brief description of your algorithm and a brief justification of its correctness.

- Show that using only a single bit of memory, we can compute whether the sum of all integers seen so far is even or odd.
- Show that we can compute whether the sum of all integers seen so far is divisible by some fixed integer  $N$  using  $O(\log N)$  bits of memory.
- Assume  $N$  is prime. Give an algorithm to check if  $N$  divides the product of all integers seen so far, using as few bits of memory as possible.
- Now let  $N$  be an arbitrary integer, and suppose we are given its prime factorization:  $N = p_1^{k_1} p_2^{k_2} \dots p_r^{k_r}$ . Give an algorithm to check whether  $N$  divides the product of all integers seen so far, using as few bits of memory as possible. Write down the number of bits your algorithm uses in terms of  $k_1, \dots, k_r$ .

**Solution:**

- Using our single bit, we maintain the sum mod 2 of the integers seen so far. In each step we simply add in the current integer  $x_i$  mod 2, so correctness follows.
- Similarly as in part (a), we simply maintain the sum of the integers seen so far mod  $N$ . This value lies in  $\{0, \dots, N - 1\}$ , so we can store it with  $\log N$  bits. The sum so far is divisible by  $N$  only when our value is 0.
- We just need a single bit to track whether or not one of the integers seen so far is divisible by  $N$ . If so, the product is divisible by  $N$ . If not, the product is not divisible by  $N$  because  $N$  is prime.
- We will use  $\lceil \log_2(k_1 + 1) \rceil + \lceil \log_2(k_2 + 1) \rceil + \dots + \lceil \log_2(k_r + 1) \rceil$  bits: for each  $i$  between 1 and  $r$ , we track the largest value  $t_i \leq k_i$  such that  $p_i^{t_i}$  divides the product of all seen integers. We start with  $t_i = 0$  for all  $i$ . When we receive a new integer  $y$ , for each  $i$  we find the largest  $t'_i$  such that  $p_i^{t'_i}$  divides  $y$ , and we set  $t_i = \min\{t'_i + t_i, k_i\}$ . We stop once  $t_i = k_i$  for all  $i$ , as then  $N$  must divide the product of all the integers we have received.

## 4 Random Elements of Streams (10 points)

- (a) Design an algorithm that takes in a stream  $z_1, \dots, z_T$  of  $T$  integers in  $[m]$  and at any time  $t$  can output a uniformly random element in  $z_1, \dots, z_t$ . Your algorithm may use at most polynomial in  $\log m$  and  $\log T$  space. Prove the correctness and analyze the space complexity of your algorithm. Your algorithm may only take a single pass of the stream.

*Hint: what is the probability that the randomly chosen element in  $z_1, \dots, z_t$  is  $z_t$ ?*

- (b) For a stream  $z_1, \dots, z_{2m}$  of  $2m$  integers in  $[m]$ , we call  $y \in [m]$  a *duplicate element* if it occurs more than once.

Design an algorithm that takes in the stream  $z_1, \dots, z_{2m}$ , and with probability at least  $1 - \frac{1}{m}$  outputs a duplicate element. Your algorithm may use at most polynomial in  $\log m$  space. Prove the correctness and analyze the space complexity of your algorithm. Your algorithm may only take a single pass of the stream.

### Solution:

- (a) We maintain a value  $x$  (using  $\log m$  bits) that is a uniformly random element from  $z_1, \dots, z_t$ , and we also store the value of  $t$  (using  $\log T$  bits). Upon receiving  $z_{t+1}$ , we know that a uniformly random element from  $z_1, \dots, z_{t+1}$  is  $z_{t+1}$  with probability  $\frac{1}{t+1}$ , and otherwise is a random element from  $z_1, \dots, z_t$ . Therefore with probability  $\frac{1}{t+1}$  we update  $x$  to equal  $z_{t+1}$ , and with probability  $\frac{t}{t+1}$  we leave  $x$  unchanged. We then increment  $t$ .
- (b) Because our stream has  $2m$  integers but only  $m$  possible values, there are at most  $m$  distinct  $t \in \{1, \dots, 2m\}$  for which  $t$  is the last time at which value  $z_t$  occurs. Thus for a random  $t$ , then  $z_t$  has a duplicate  $z_{t'} = z_t$  at some later time  $t' > t$  with probability at least  $\frac{1}{2}$ .

Therefore we can randomly choose a time  $t$  (up front, before starting the stream), and then store  $z_t$  at time  $t$  and check if any subsequent integer we see equals  $z_t$ . As shown above, we find a duplicate with probability  $\geq \frac{1}{2}$ . This scheme requires  $O(\log m)$  space to store  $t$  and  $z_t$ .

To boost the success probability, we simply run  $\log m$  copies of the scheme above, so that the chance we fail to find a duplicate is at most  $(\frac{1}{2})^{\log m} = \frac{1}{m}$ . The space usage then becomes  $O(\log m)^2$ .

## 5 Streaming Majority (10 points)

Design a deterministic algorithm that takes in a stream  $z_1, \dots, z_m$  of integers in  $[m]$ , and outputs the most common value  $y$  in the stream, assuming  $y$  occurs more than  $m/2$  times. If  $y$  occurs at most  $m/2$  times, your algorithm may output anything. Your algorithm should use at most polynomial in  $\log m$  space.

**Solution:** Maintain a candidate  $x$  and a counter  $c$ , initially with  $c = 0$ . For each stream element  $z_t$ , do the following:

- If  $c = 0$ , set  $x = z_t$  and  $c = 1$ .
- Else if  $z_t = x$ , increment  $c$ .
- Else decrement  $c$ .

At the end, output  $x$ .

The algorithm uses  $O(\log m)$  bits:  $O(\log m)$  bits for  $x \in [m]$  and  $O(\log m)$  bits for the counter  $c \leq m$ .

For correctness, view each decrement as canceling one copy of the current candidate with one different element. Thus the algorithm repeatedly removes pairs of distinct elements. Removing such pairs cannot destroy a strict majority element: if some  $y$  appears more than  $m/2$  times, then after any number of cancellations,  $y$  still appears more often than all other remaining values combined. Hence some copy of  $y$  remains unmatched, so the final candidate must be  $y$ , as the final candidate is the only value with unmatched copies at the end. If no value appears more than  $m/2$  times, the algorithm may output anything.