

# CSCI 7000: Advanced Data Structures (Assignment 1)

Nicholas Papadopoulos

September 8, 2022

## Problem 1

### Part A

$$\begin{aligned}x_1 \oplus x_2 &= x_1 \oplus x_3 \\x_1 \oplus x_1 \oplus x_2 &= x_1 \oplus x_1 \oplus x_3 \\x_2 &= x_3\end{aligned}$$

So, we need to find the probability that  $x_2 = x_3$ . There are  $2^l$  possible bitstrings of length  $l$ , so the probability that the same bitstring is randomly generated for both  $x_2$  and  $x_3$  is  $\frac{1}{2^l}$ .

### Part B

Set  $x_1 \oplus x_2$  can map to some other random string, say  $s$ . Each bitstring,  $s$ , of length  $l$  can have  $2^l$  pairs, where order matters, that xor to  $s$ . This is because you can choose any random bitstring,  $r$ , of length  $l$ , xor it with  $s$ , and the result,  $r'$  will be the paired value of  $r$  where  $r \oplus r' = s$ . In other words, any random string out of the possible  $2^l$  strings can serve as  $x_3$  with a deterministic, corresponding  $x_4$ .

Hence,  $x_1$ ,  $x_2$ , and  $x_3$  can be anything without consequence. With these set, however, we know that  $x_4 = x_1 \oplus x_2 \oplus x_3$ . In other words, there is only one possible solution for  $x_4$  once the free variables  $x_1$ ,  $x_2$ , and  $x_3$  are chosen. Since  $x_4$  is chosen randomly, and there are  $2^l$  possible choices, the probability that  $x_1 \oplus x_2 = x_3 \oplus x_4$  is  $\frac{1}{2^l}$ .

### Part C

First, we can determine the probability that  $x$  maps to the same keys for  $A$  and  $B$ . That is, what is the probability that  $x_1 = x_2$ ? We have determined that this probability is  $\frac{1}{2^l}$  in part (A).

$$\begin{aligned}h_{A,B}(x) &= A[x_1] \oplus B[x_1] \\h_{A,B}(y) &= A[y_1] \oplus B[y_1]\end{aligned}$$

We can now ask what the probability is that  $A[x_1] \oplus B[x_1] = A[y_1] \oplus B[y_1]$ . Since  $A[x_1]$ ,  $A[x_2]$ ,  $A[y_1]$ , and  $A[y_2]$  are random  $l$ -bit strings, we can use our answer in part (B) to say that the probability of collision is  $\frac{1}{2^l}$ .

## Problem 2

$$\begin{aligned}h_1(x) &= (x \bmod 6) \bmod 4 \\h_2(x) &= (2x \bmod 6) \bmod 4 \\h_3(x) &= (3x \bmod 6) \bmod 4 \\h_4(x) &= (4x \bmod 6) \bmod 4 \\h_5(x) &= (5x \bmod 6) \bmod 4\end{aligned}$$

Here, we can see that we can pick  $h_3$  and see that

$$h_3(x) = \begin{cases} 0 & x \text{ is even} \\ 3 & x \text{ is odd} \end{cases}$$

This is because 3 times any even number will be exactly divisible by 6, giving a remainder of 0, and 3 times and odd number will be 3 times and even number plus 3, giving a remainder of 3. Since both 0 and 3 are less than 4, modding by 4 does not effect the result. Hence,  $\frac{1}{2}$  off the possible keys to  $h_3$  will result in the same answer, so there the probability of a collision is  $\frac{1}{2} > \frac{1}{4}$ .

## Problem 3

We can take the binary representation of each character and mod them by some number, say 10. Then we can multiply each character by subsequent powers of ten so that no character can interfere with the next when added together, since the highest value of some power,  $p$ ,  $9 * 10^p < 10^{p+1}$ . This holds true for any base, say  $a$ , not only 10. In order to ensure some value for each character, we can mod by  $a - 1$  and add 1 to the result. We can use this fact to include a salt, so we can say that

$$a = b + \text{salt}$$

Where  $b$  is some minimum base number we would like to utilize. We can then define a character encoding by

$$(x_n = c_n \bmod (a - 1)) + 1$$

We then have that

$$h(s) = x_1(a^n) + x_2(a^{n-1}) + \dots x_n(a^0)$$

$h(t)$  would then be

$$h(t) = x_2(a^n) + x_3(a^{n-1}) + \dots x_{n+1}(a^0)$$

So, to retrieve  $h(t)$  given  $s$ ,  $t$ , and  $h(s)$ , we simply subtract  $x_1(a^n)$ , multiply the result by  $a$ , then add  $x_{n+1}$ .

$$h(t) = (h(s) - x_1(a^n))a + x_{n+1}$$

This uses constant time operations for all portions of this equation, consisting of modulo, exponentiation, subtraction, multiplication, and addition.

## Problem 4

### Explanation

My solution first implements the naive version, where all substrings are stored in a hash table, with the key being the substring itself and the value being a list of indices where that substring was found. It begins at the first index of the first word, extracts the substring from that index of the given length, then inserts the substring into the hash along with the start index. It will then move to the next index and repeat until all substrings of the word have been stored in the set. It then repeats this process with the second word in a second hash table. Finally, it goes through each entry in one of the hash tables and determines if there is a matching substring in the second hash table by querying the key. If there is a matching key, it creates a set product of the two words' index lists, creating all possible combinations of start indices for both words that generate the same substring.

The more complex solution actually uses nearly the same algorithm, except it hashes the keys using the process described in problem 3 of this homework instead of using the substring

directly. So, instead of extracting each substring at each start index for a word, it first hashes only the first substring. Then, it hashes subsequent keys by giving only the next character and the first character of the hash before it.