15-451/651 Algorithm Design & Analysis, Fall 2023

Recitation #2

Objectives

- Learn about k-wise independent hashing and practice proving it
- Understand the technique of *fingerprinting* and apply it to solve string problems

Recitation Problems

1. **(k-wise Independent Hashing)** Recall from class that a hash family $\mathcal{H}$ is k*-wise independent* if for all k distinct keys $x_1, x_2, \ldots, x_k \in \mathcal{U}$ and every set of k values $v_1, v_2, \ldots, v_k \in \{0, 1, \ldots, m-1\}$, we have that

$$\Pr_{h \in \mathcal{H}}[h(x_1) = v_1 \wedge h(x_2) = v_2 \wedge \cdots \wedge h(x_k) = v_k] = \frac{1}{m^k}$$

Intuitively, this means that if you look at only up to k keys, the hash family appears to hash them truly randomly.

(a) Is this hash family from $U = \{a, b\}$ to $\{0, 1\}$ (i.e., $m = 2$) universal? one-wise independent (i.e., uniform)? how about pairwise independent?

	a	b
h_1	0	0
h_2	1	0

(b) Can you fill in the blanks in this hash family with values in $\{0, 1\}$ to make it pairwise independent? 3-wise independent?

	a	b	c
h_1	0	0	
h_2			1
h_3	0		
h_4	1	1	0

2. **(Extended Matrix Method)** In lecture we covered the *matrix method* of hashing integers by interpreting them as binary vectors from the universe $\mathcal{U} = \{0, 1\}^w$, into a table of size $m = 2^b$ indexed by $\{0, 1\}^b$ where each hash function in the family is defined by a random matrix $A \in \{0, 1\}^{b \times w}$ and

$$h(x) = Ax \mod 2$$

(a) Prove that the matrix method is not 1-wise independent (i.e., not uniform).

(b) Now suppose we extend the matrix method to be defined as

$$h(x) = Ax + c \mod 2$$

where $c \in \{0,1\}^b$ is a random binary vector.

Prove that this extension of the matrix method is pairwise independent.

3. **(A String Matching Oracle)** In this recitation we generalize the fingerprinting method described in lecture. Let $T = t_0, t_1, \ldots, t_{n-1}$, be a string over some alphabet $\Sigma = \{0, 1, \ldots, z-1\}$. Let $T_{i,j}$ denote the substring $t_i, t_{i+1}, \ldots, t_{j-1}$. This string is of length $j - i$. We want to preprocess T such that the following comparison of two substrings of T of length ℓ can be answered (with a low probability of a false positive) in constant time:

$$\text{Test if } T_{i,i+\ell} = T_{j,j+\ell}$$

First of all let's define the fingerprinting function. Let p be a prime, along with a base b (larger than the alphabet size). The Karp-Rabin fingerprint of T is

$$h(T) = (t_0 b^{n-1} + t_1 b^{n-2} + \cdots + t_{n-1} b^0) \bmod p$$

From now on we will omit the mod p from these expressions.

Now, to preprocess the string T, we will compute the following arrays for $0 \le i \le n$: ($\textit{Don't forget we are omitting the}$ mod*s!*)

$$\begin{aligned} r[i] &= b^i \\ a[i] &= t_0 b^{i-1} + t_1 b^{i-2} + \cdots + t_{i-1} b^0 \end{aligned}$$

(a) Give algorithms for computing these in time $O(n)$:

(b) Find an expression for $h(T_{i,j})$.

So the end result is that we can test if $T_{i,i+\ell} = T_{j,j+\ell}$ by comparing $h(T_{i,i+\ell})$ with $h(T_{j,j+\ell})$. The probability of a false positive can be made as small as desired by picking a sufficiently large random prime p, as seen in lecture. (Here we are not concerned with bounding the false positive probability.)

4. **(Optional: Palindrome Counting)** Given a text $T \in \Sigma^n$ represented as an array of characters, devise an algorithm to count the number of substrings of T that are palindromes in $O(n \log n)$ time with error rate at most some given $\epsilon > 0$.

   (a) Find a way to check in $O(1)$ if a given substring $T_{i,j}$ is a palindrome (with high probability). You can use $O(n)$ preprocessing time.

(b) Now, find the number of palindromic substrings of T. (Hint: If $T_{i,j}$ is a palindrome, then what other substrings do we know are palindromes?)

Fun fact: This problem can be solved deterministically (without hashing) in $O(n)$ using Manacher's Algorithm.