Hands-On 2: Algorithm Design A.Y. 2024

Giovanni Braccini Sounak Mukhopadhyay
Giacomo Trapani

25 March 2024

1 Exercise 1

Consider the Karp-Rabin string matching in its Las Vegas version, where the fingerprint $f(x) = x \mod p$ *for a random prime* p *in* $[2...\tau]$ *is replaced by* $h(x) = (ax + b \mod p) \mod q$*, where now* $p > q$ *are given primes, and* $a \neq 0$ *and* b *are random integers from* $[0...p-1]$

Exercise 1.a

Discuss whether h(x) can be still computed as a rolling hash, and which are the benefits of replacing f(x) with h(x).

Denoting with $asc(x)$ the function which converts a letter to its ASCII encoding, we encode the text with the function

$$enc(x_0, \ldots, x_i) = \prod_{j=0}^{i} asc(x_j)$$

The function h is a rolling hash function, it is possible to calculate a new hash value starting from the old hash value, the old value removed from the window, and the new value added to the window:

1. Starting from a window made up of values $(x_0, \ldots, x_i)$, we compute the value $y = h(x_0, \ldots, x_i)$.
2. We remove the old value x_0 computing $y \cdot x_0^{-1} \mod p$.
3. We add the new value x_{i+1} computing $y \cdot x_{i+1} \mod p \mod q$.

This reasoning can be applied iteratively throughout the whole text we need to parse.

The benefits in using h over f are:

- h is 2-independent;
- we can pick two values a, b to cause as few collisions as possible.

The drawbacks are:

- using f we can make the collision chance arbitrarily small (it has collision chance $1/n^c$), while h has a fixed collision chance;
- generating random primes p, q has a cost.

Exercise 1.b

Does the expected running time change significantly?
No, the running time is the same as in using f.

2 Exercise 2

Your company has a database $S \subseteq U$ of keys. For this database, it uses a hash function h uniformly chosen at random from a universal family H (as seen in class). It also keeps a bit vector B of m entries, initialized to zeroes, which are then set $B[h(k)] = 1$ for every $k \in S$ (note that collisions may happen). Unfortunately, the database S has been lost, thus only B and h survived, and the rest is no more accessible.

Exercise 2.a

Under the hypothesis that $m \geq c|S|$ for some $c > 1$, can we use the expected number of 1s in B to estimate $|S|$ under a uniform choice at random of $h \in H$?
We call $n = |S|$ and k the number of hash functions used, which in this case we know to be 1.

We define m random indicator variables s.t.

$$X_i = \begin{cases} 1 \text{ if } B[i] = 1 \\ 0 \text{ otherwise} \end{cases}$$

We denote with y the number of bits set to 1 in the bloom filter, and we know its value $y = \sum_{i=1}^{m} B[i]$.

Denoting with p the probability of a bit in the bloom filter to be set to 1, we can compute the sum of expected values of X_i:

$$E[\sum_{i=1}^{m} X_i] = \sum_{i=1}^{m} X_i E[X_i] = m \cdot p$$

From bloom filter theory, we know we can compute the value of p with the formula

$$p = \left(1 - \left(1 - \frac{1}{m}\right)^{k \cdot n}\right)^k = 1 - \left(1 - \frac{1}{m}\right)^n = 1 - \left(1 - \frac{1}{m}\right)^{mn/m}$$

We assume m is large enough for the following identity to prove true

$$\lim_{x\to\infty}\left(1-\frac{1}{x}\right)^x=\frac{1}{e}$$

This means we can write p as

$$p=1-\left(\frac{1}{e}\right)^{n/m}$$

We can put everything together and derive a formula for n:

$$\begin{gathered}
y=m\cdot p\approx m\cdot\left(1-e^{-n/m}\right)\\
\frac{y}{m}=\left(1-e^{-n/m}\right)\\
\frac{y}{m}-1=\left(-e^{-n/m}\right)\\
\log\left(1-\frac{y}{m}\right)=-\frac{n}{m}\\
n=-m\cdot\log\left(1-\frac{y}{m}\right)
\end{gathered}$$

$\square$

Exercise 2.b

Based alone on B and h, how can you establish if any key given $k \in U$ was also in S or not, and what is the probability of error?

We can first compute the hash of the given key $y = h(k)$. We distinguish two cases:

1. **The bit in position $B[y]$ is** 0. We are certain k was not inserted i.e. $k \notin S$.

2. **The bit in position $B[y]$ is** 1. In this case, either the key was inserted or we have a false positive. We know from bloom filter theory using only 1 hash function, the probability of a false positive is $1 - (1 - p)^n$, with p the probability of a specific bit to be set to 1.

   Furthermore, we know n from *Exercise 2.a* and, from hash function theory we have

   $$Pr[h(x) = y] = \frac{1}{m}$$

   This means we can write the probability of an error as

   $$\Pr\left[\text{error}\right] = 1 - \left(1 - \frac{1}{m}\right)^n$$