15-451/651: Design & Analysis of Algorithms September 21, 2017
Lecture #8: Fingerprinting and String Matching last changed: September 19, 2017

In today's lecture, we will talk about randomization and hashing in a slightly different way. In particular, we use arithmetic modulo prime numbers to (approximately) check if two strings are equal to each other. Building on that, we will get an randomized algorithm (called the *Karp-Rabin fingerprinting scheme*) for checking if a long text T contains a certain pattern string P as a substring.

1 How to Pick a Random Prime

In this lecture, we will often be picking random primes, so let's talk about that. (In fact, you do this when generating RSA public/private key pairs.)

How to pick a random prime in some range $\{1, \ldots, M\}$? The answer is easy.

- Pick a random integer X in the range $\{1, \ldots, M\}$.
- Check if X is a prime. If so, output it. Else go back to the first step.

How would you pick a random number in the prescribed range? Also easy. Pick a uniformly random bit string of length $\lfloor \log_2 M \rfloor + 1$. (We assume we have access to a source of random bits.) If it represents a number $\leq M$, output it, else repeat. The chance that you will get a number $\leq M$ is at least half, so in expectation you have to repeat this process at most twice.

How do you check if X is prime? You can use the Miller-Rabin randomized primality test (which may err, but it will only output "prime" when the number is composite with very low probability). There are other randomized primality tests as well, see the Wikipedia page. Or you can use the Agrawal-Kayal-Saxena primality test, which has a worse runtime, but is deterministic and hence guaranteed to be correct.

2 How Many Primes?

You have probably seen a proof that there are infinitely many primes. Here's a different question that we'll need for this lecture.

For positive integer n, how many primes are there in the set $\{1, 2, \ldots, n\}$?

Let there be $\pi(x)$ primes between 0 and x. One of the great theorems of the 20^{th} century was the Prime Number theorem which proved that

$$\lim_{n \to \infty} \frac{\pi(x)}{x/(\ln x)} = 1.$$

And while this is just a limiting statement, an older result of Chebyshev (from 1848) says that for $n \geq 2$,

$$\pi(n) \geq \frac{7}{8} \frac{n}{\ln n} = (1.262\ldots) \frac{n}{\lg n} > \frac{n}{\lg n}$$

Here are two consequences of this theorem. The first is that a random integer between 1 and n is a prime number with probability at least $\frac{1}{\lg n}$. Put another way, we also get the following useful fact:

Fact 1 *If we want at least $k \geq 4$ primes between 1 and n, it suffices to have $n \geq 2k \lg k$.*

Proof: Just plugging in, we get $\pi(2k \ln k) \geq \frac{2k \ln k}{\lg(2k \lg k)} \geq \frac{2k \ln k}{\lg 2 + \lg k + \lg \lg k} \geq k$. ■

2.1 Tighter Bounds

The following even tighter set of bounds were proved by Pierre Dusart in 2010. For all $x \geq 60184$ we have:

$$\frac{x}{\ln x - 1.1} > \pi(x) > \frac{x}{\ln x - 1}$$

Because this is a two-sided bound, it allows us to deduce a lower bound on the number of primes in a range. For example, the number of 9-digit prime numbers (i.e. primes in the range $[10^8, 10^9 - 1]$) is

$$\pi(10^9 - 1) - \pi(10^8 - 1) > \frac{10^9 - 1}{\ln(10^9 - 1) - 1} - \frac{10^8 - 1}{\ln(10^8 - 1) - 1.1} = 44928097.3\ldots$$

From this we can infer that a randomly generated 9 digit number is prime with probability at least $0.049920\ldots$. Thus, the random sampling method would take at most 21 iterations in expectation to find a 9-digit prime.

3 The String Equality Problem

Here's a simple problem: we're sending a Mars lander. Alice, the captain of the Mars lander, receives an N-bit string x. Bob, back at mission control, receives an N-bit string y. Alice knows nothing about y, and Bob knows nothing about x. They want to check if the two strings are the same, i.e., if $x = y$. [1]

One way is for Alice to send the entire N-bit string to Bob. But N is very large. And communication is super-expensive between the two of them. So sending N bits will cost a lot. *Can Alice and Bob share less communication and check equality?*

If they want to be 100% sure that $x = y$, then one can show that fewer than N bits of communication between them will not suffice. But suppose we are OK with being correct with probability 0.9999. Formally, we want a way for Alice and Bob to send a message to Bob so that, at the end of the communication:

- If $x = y$, then Pr[Bob says `equal`] $= 1$.
- If $x \neq y$, then Pr[Bob says `unequal`] $= 1 - \delta$.

Here's a protocol that does almost that:

1. Alice picks a random prime p from the set $\{1, 2, \ldots, M\}$ for $M = \lceil 2 \cdot (5N) \cdot \lg(5N) \rceil$.
2. She sends Bob the prime p, and also the value $h_p(x) := x \mod p$.
3. Bob checks if $h_p(x) = y \mod p$. If so, he says `equal` else he says `unequal`.

[1] E.g., this could be the latest update to the lander firmware, and we want to make pretty sure the file did not get corrupted in transition.

For now, let's not worry about where the particular value of M came from: it will arise naturally. Let's see how this protocol performs.

Lemma 2 *If $x = y$, then Bob always says* `equal`*.*

Proof: Indeed, if $x = y$, then $x \mod p = y \mod p$. So Bob's test will always succeed. ■

Lemma 3 *If $x \neq y$, then* $\Pr[$ *Bob says* `equal` $] \leq 0.2$.

Proof: Consider x and y as N-bit binary numbers. So $x, y < 2^N$. Let $D = |x - y|$ be their difference. Bob says `equal` only when $x \mod p = y \mod p$, or equivalently $(x - y) = 0 \mod p$. This means p divides $D = |x - y|$. In words, the random prime p we picked happened to be a divider of D. What are the changes of that? Let's do the math.

The difference D is a N-bit integer, so $D \leq 2^N$. So D can be written (uniquely) as $D = p_1 p_2 \cdots p_k$, each p_i being a prime, where some of the primes might repeat[2]. Each prime $p_i \geq 2$, so $D = p_1 p_2 \cdots p_k \geq 2^k$. Hence $k \leq N$. The difference D has at most N divisors. The probability that p is one of them is

$$\frac{N}{\text{number of primes in } \{1, 2, \ldots, M\}}.$$

We want this to be at most 1/5. I.e., we would like that the number of primes in $\{1, 2, \ldots, M\}$ is at least $5N$. But Fact 1 says that choosing $M \geq 10N \lg 5N$ will give us at least $5N$ primes. Hence

$$\Pr[\text{ Bob says } \texttt{equal} \text{ and hence errs }] \leq \frac{N}{\text{number of primes in } \{1, 2, \ldots, M\}} \leq \frac{N}{5N} \leq \frac{1}{5}.$$

■

3.1 Reducing the Error Probability

If you don't like the probability of error being 20%, there're two ways to reduce the probability of error.

Approach #1: Have Alice choose a random prime from a larger set. For some integer $s \geq 1$, if we choose $M = 2 \cdot sN \lg(sN)$, then the arguments above show that the number of primes in $\{1, \ldots, M\}$ is at least sN. And hence the probability of error is $1/s$. Now choose any s large enough.

Approach #2: Just have Alice repeat this process 10 times independently, with Bob saying `equal` if and only if all in 10 repetitions, the test passes. The chance that he will make an error (i.e., say `equal` when $x \neq y$) is only

$$(1/5)^{10} = \frac{1024}{10^{10}} \leq 0.000001.$$

In general, if we repeat R times, we get the probability of error is at most

$$(1/5)^R.$$

3.2 Why did Alice not just send x over to Bob?

Naïvely, Alice could have sent x over to Bob. That would take N bits. Now she sends the prime p, and $x \mod p$. That's two numbers at most $M = 10N \lg 5N$. The number of bits required for that: $2 \log M = 2 \log(10N \lg 5N)) = O(\log N)$.

[2] This unique prime-factorization theorem is known as the fundamental theorem of arithmetic.

To put this in perspective, suppose x and y were two copies of all of Wikipedia. Say that's about 25 billion characters. Say 8 bits per character, so $N \approx 2 \cdot 10^{11} \approx 2^{38}$ bits. Whereas our approach, even with repeating things 10 times, sent over $10 \cdot 2 \log(10N \lg 5N)) \leq 1280$ bits. That's a lot less communication!

4 The Karp-Rabin Algorithm (a.k.a. the "Fingerprint" Method)

Let's use this idea to solve a different problem. In the *string matching* problem, we are given

- A *text* T, of length m.
- A *pattern* P, of length n.

The goal is to output all the occurrences of the pattern P inside the text T. E.g., if $T =$ `abracadabra` and $P =$ `ab` then the output should be $\{0, 7\}$.

There are many ways to solve this problem, but today we will use randomization to solve this problem. This solution is due to Karp and Rabin.[3] The idea is smart but simple, elegant and effective—like in many great algorithms.

4.1 The Karp-Rabin Idea

Think about the hash function $h_p(x) = x \mod p$, for $x \in \{0,1\}^n$. Now look at the string x' obtained by dropping the leftmost bit of x, and adding a bit to the right end. E.g., if $x =$ `0011001` then x' might be `0110010` or `0110011`. If I told you $h_p(x) = z$, can you compute $h_p(x')$ fast?

Suppose x'_{lb} is the lowest-order bit of x', and x_{hb} be the highest order bit of x. Then

$$\text{value of } x' = 2(\text{value of } x - x_{hb} \cdot 2^{n-1}) + x'_{lb}$$

Remember that $h_p(a+b) = (h_p(a) + h_p(b)) \mod p$, and $h_p(2a) = 2h_p(a) \mod p$. So

$$h_p(x') = (2h_p(x) - x_{hb} \cdot h_p(2^n) + x'_{lb}) \mod p.$$

That's two function computations to compute $h_p(x)$ and $h_p(2^n)$, three arithmetic operations modulo p, and one more residue modulo p.

4.2 How to use this idea for String Matching

To keep things short, let $T_{a \ldots b}$ denote the string from the a^{th} to the b^{th} positions of T, inclusive. So the string matching problem is: output all the locations $a \in \{0, 1, \ldots, m-n\}$ such that

$$T_{a \ldots a+(n-1)} = P.$$

Here's the algorithm:

1. Pick a random prime p in the range $\{1, \ldots, M\}$ for a $M = \lceil 2sn \lg(sn) \rceil$ for a value of s we'll choose later.

[3] Again, familiar names. Dick Karp is a professor of computer science at Berkeley, and won the Turing award in 1985. Among other things, he developed two of the max-flow algorithms you saw, and his 1972 paper showed that many natural algorithmic problems were NP complete. Michael Rabin is professor at Harvard; he won the Turing award in 1976 (jointly with CMU's Dana Scott). You may know him from the popular Miller-Rabin randomized primality test (the Miller there is our own Gary Miller); he's responsible for many algorithms in cryptography.

2. Compute $h_p(P)$ and $h_p(2^n)$, and store these results.
3. Compute $h_p(T_{0 \ldots n-1})$, and check if it equals $h_p(P)$. If so, output `match at location 0`.
4. For each $i \in \{0, \ldots, m-n\}$, compute $h_p(T_{i+1 \ldots i+n})$ using $h_p(T_{i \ldots i+n-1})$.
   If $h_p(T_{i+1 \ldots i+n}) = h_p(P)$, output `match at location` $i+1$.

Clearly, we'll never have a false negative (i.e., miss a match) but we may erroneously output location that are not matches (have a false positive). Let's analyze the error probability, and the runtime.

4.2.1 Probability of Error

We do m different comparisons, each has a probability $1/s$ of failure. So, by a union bound, the probability of having at least one false positive is m/s. Hence, setting $s = 100m$ will make sure we have a $\frac{1}{100}$ chance of even a single mistake.

This means we set $M = (200 \cdot mn) \lg(200 \cdot mn)$. Which requires $\leq \lg M + 1 = O(\log m + \log n)$ bits to store. And hence our prime p is also at most $O(\log m + \log n)$ bits.[4]

4.2.2 Running Time

Let's say we can do arithmetic and comparisons on $O(\log M)$-bit numbers in constant time. (See the footnote at the bottom of this page about why this is reasonable.) And let's not worry about the time to pick a random prime for now.

- Computing $h_p(x)$ for n-bit x can be done in $O(n)$ time. So each of the hash function computations in Steps 2 and 3 take $O(n)$ time.
- Now, using the idea in Section 4.1, we can compute each subsequent hash value in $O(1)$ time! So iterating over all the values of i takes $O(m)$ time.

That's a total of $O(m+n)$ time! And you can't do much faster, since the input itself is $O(m+n)$ bits long.

4.3 Extensions and Connections

There are other (deterministic) fast ways of solving the string matching problem we mentioned above. See, e.g., the Knuth-Morris-Pratt algorithm, and suffix trees. The advantage of the Karp-Rabin approach is not only the simplicity, but also the extendability. You can, e.g., solve the following 2-dimensional problem using the same idea:

> **2-dimensional pattern matching.** Given a $m_1 \times m_2$-bit rectangular text T, and a $n_1 \times n_2$-bit pattern P (where $n_i \leq m_i$), find all occurrences of P inside T. Show that you do this in $O(m_1 m_2)$ time, where we assume that you can do modular arithmetic with integers of value at most poly$(m_1 m_2)$ in constant time.

[4] If you do the math, and say $m, n \geq 10$, then $\lg M \leq 4(\lg m + \lg n)$. Now, just for perspective, if we were looking for a $n = 1024$-bit phrase in Wikipedia, this means the prime p is only $4(\lg 2^{38} + \lg 2^{10}) \leq 192$ bits long.