Lecture Notes on Fingerprinting & Applications

May 3, 2007

Fingerprinting

Suppose we have two copies of a large database in locations that are connected by an expensive and error-prone communication link. We wish to test the integrity of the copies, i.e., to check that both are the same. The obvious solution of sending one copy of the database across the link and doing a direct comparision is ruled out because of cost and reliability issues. In such situations, we would like to shrink our data down to a much smaller *fingerprint* which is easier to transmit. Of course, this is useful only if the fingerprints of different pieces of data are unlikely to be the same.

We model this situation with two spatially separated parties, Alice and Bob, each of whom holds an n-bit number (where n is very large). Alice's number is $a = a_1 a_2 \dots a_n$ and Bob's is $b = b_1 b_2 \dots b_n$. Our goal is to decide if $a = b$ without transmitting all n bits of the numbers between the parties.

The Protocol:

Alice picks a prime number u.a.r. from the set $\{2, \dots, T\}$, where T is a value to be determined. She computes her fingerprint as $F_p(a) = a \bmod p$. She then sends p and $F_p(a)$ to Bob. Using p, Bob computes $F_p(b) = b \bmod p$ and checks whether $F_p(b) = F_p(a)$. If not he concludes that $a \neq b$, else he presumes that $a = b$.

Observe that if $a = b$ then Bob will always be correct. However, if $b \neq a$ then there may be an error: this happens iff the fingerprints of a and b happen to coincide. We now show that, even for a modest value of T (exponentially smaller than a and b), if $a \neq b$ then $\Pr[F_p(a) = F_p(b)]$ is small.

First observe that, if $F_p(a) = F_p(b)$, then $a = b \bmod p$, so p must divide $|a - b|$. But $|a - b|$ is an n-bit number, so the number of primes p that divide it is (crudely) at most n (because each prime is at least 2). Thus the probability of error is at most $\frac{n}{\pi(T)}$, where $\pi(x)$ is defined as the number of primes less than or equal to x.

We now appeal to a standard result in Number Theory:

Prime Number Theorem:

$$\pi(x) \sim \frac{x}{\ln x} \text{ as } x \to \infty.$$

Moreover,

$$\frac{x}{\ln x} \leq \pi(x) \leq 1.26 \frac{x}{\ln x} \quad \forall x \geq 17.$$

Thus we may conclude that

$$\Pr[\text{error}] \leq \frac{n}{\pi(T)} \leq \frac{n \ln T}{T}.$$

Setting $T = cn \ln n$ for a constant c gives

$$\Pr[\text{error}] \leq \frac{n(\ln n + \ln \ln n + \ln c)}{cn \ln n} = \frac{1}{c} + o(1).$$

Actually, we can improve this analysis slightly, using another fact from Number Theory: the number of primes that divide any given n-bit number is at most $\pi(n)$ (which is a factor $\ln n$ smaller than the very crude bound of n we used above). Thus we get the improved error bound

$$\Pr[\text{error}] \leq \frac{\pi(n)}{\pi(T)} \leq 1.26 \frac{n \ln T}{T \ln n}.$$

Setting $T = cn$ now gives us an error probability of only

$$\frac{1.26}{c}\left(1 + \frac{\ln c}{\ln n}\right),$$

which is small even for modest c.

Since this algorithm has one-sided error, the error probability can be reduced as usual by repeated trials, i.e., sending multiple independent fingerprints.

The numbers transmitted in the above protocol are integers mod p, where $p = O(n)$. Hence the number of bits transmitted is only $O(\log n)$, an exponential improvement over the deterministic scenario!

Example: If $n = 2^{23}$ (≈ 1 megabyte) and $T = 2^{32}$ (so that fingerprints are the size of a 32-bit word), then

$$\Pr[\text{error}] \leq 1.26 \frac{n \ln T}{T \ln n} = 1.26 \cdot \frac{2^{23}}{2^{32}} \cdot \frac{32}{23} < 0.0035.$$

The above protocol requires us to pick a random prime in $\{2, \ldots, T\}$. A simple algorithm for this is to pick a random number in the interval and check if it is a prime; if so, output it, else try again. The Prime Number Theorem tells us that the probability of success in such a trial is at least $\frac{1}{\ln T}$, so the expected number of trials until we find a prime is at most $\ln T$, which is small. (Recall that $\ln T = O(\log n)$.) In the next lecture, we shall see how to check efficiently whether a given number is a prime; this will actually involve another important use of randomization.

Applications of Fingerprinting

1. Arithmetic modulo a random prime

In many applications, we need to do arithmetic on very large integers (so large that precision is lost or overflow occurs). For example, these can occur as a result of evaluating a polynomial or a determinant. Frequently all we need to know is whether two such large integers are equal or not (e.g., does a certain polynomial evaluate to zero at some given point?) We can deal with this problem using fingerprints in a very simple way: just do all the arithmetic modulo a random prime p. From our analysis above we know that, if the integers have n bits, then the number of bits needed in p to ensure a correct answer with high probability is only $O(\log n)$, which is exponentially less and probably small enough to avoid loss of precision.

2. Pattern matching

Suppose we have a long source text $X = x_1 x_2 ... x_n$ and a shorter pattern text $Y = y_1 y_2 ... y_m$ with $m < n$. We would like to determine whether or not Y occurs as a contiguous substring of X, i.e., whether $Y = X(j) \equiv x_j x_{j+1} ... x_{j+m-1}$ for some j.

The standard deterministic algorithm that compares the pattern to the source text bit-by-bit until a match is found or the end of the source is reached clearly runs in $O(mn)$ time.

There are complicated deterministic algorithms due to Boyer/Moore and Knuth/Morris/Pratt that run in $O(m + n)$ time, but they are difficult to implement and have a rather large overhead.

We now present a very simple and practical randomized algorithm, due to Karp and Rabin, that also runs in $O(m+n)$ time. This algorithm computes a fingerprint of Y, and compares it to the fingerprints of successive substrings of X.

```
pick a random prime $p \in \{2, ..., T\}$
compute $F_p(Y) = Y \bmod p$
**for** $j = 1$ to $n - m + 1$ **do**
    compute $F_p(X(j))$
    **if** $F_p(Y) = F_p(X(j))$
    **then** output "match?" and halt
output "no match!"
```

Error probability

This algorithm has one-sided error: it may output "match" when there is in fact no match. Following the analysis of fingerprinting above, a simple upper bound on $\Pr[\text{Error}]$ is $n\frac{\pi(m)}{\pi(T)}$; this follows from the probability of error in the comparison between Y and each $X(j)$ (which is $\frac{\pi(m)}{\pi(T)}$), together with a union bound over j. We can do better by observing that, in order for a false match to occur somewhere along the string, p must divide $|Y - X(j)|$ for some j, and therefore p must divide $\prod_j |Y - X(j)|$, which is an mn-bit number. So, the bound on the error can be improved to

$$\Pr[\text{Error}] \leq \frac{\pi(mn)}{\pi(T)}.$$

Thus, as above, if we choose $T = cmn$ for a reasonable constant c we will get a small error probability.

Running Time

To find the running time of the algorithm, we first note that p has only $O(\log(mn)) = O(\log n)$ bits, so we may reasonably assume that arithmetic mod p can be performed in constant time. First, the algorithm computes $F_p(Y)$; since Y is an m-bit number, this requires $O(m)$ time.

Next, we note that $X(j)$ and $X(j+1)$ differ in only the first and last bits, so we have the following relationship:

$$X(j+1) = 2(X(j) - 2^{m-1}x_j) + x_{j+m}.$$

The fingerprint of $X(j+1)$ can thus be computed as follows:

$$F_p(X(j+1)) = 2(F_p(X(j)) - 2^{m-1}x_j) + x_{j+m} \bmod p.$$

This involves a constant number of arithmetic operations mod p, and hence takes constant time. The loop iterates n times, so the total running time is $O(m+n)$, as claimed earlier.

Note that without the observation that we can compute $F_p(X_{j+1})$ efficiently from $F_p(X(j))$, the running time would be $O(mn)$, which is no improvement on the naive algorithm!

Note also that this algorithm can be converted into a Las Vegas one (i.e., zero error probability but with only *expected* running time $O(n+m)$) by *checking* that a match is correct before outputting it. Conceivably this could require $O(mn)$ time in the worst case (if we were unlucky enough to choose a prime that gave a huge number of false matches, because each match costs $O(m)$ to check), but the expected running time is still $O(n+m)$.

Finally, we give a numerical example that might occur (e.g.) when searching for a pattern in a string of biological data such as DNA. Take $n = 2^{12}$, $m = 2^8$, and T to be the machine word size, 2^{32}. Then we have

$$\Pr[\text{Error}] \leq \frac{\pi(mn)}{\pi(T)} \leq 1.26\frac{mn}{\ln mn}\frac{\ln T}{T} = 1.26\frac{2^{20}}{20}\frac{32}{2^{32}} \approx 0.0005.$$