Algorithms Exercise Sheet: Randomized Fingerprinting

Instructions

In this assignment, you will implement and experiment with the fingerprinting scheme we worked on in class.

Fingerprinting Scheme

This is a brief review of the scheme we discussed in Lectures 20 and 21 in class. Alice has an n-bit binary number $x = x_1x_2 \dots x_n$ (x_i is the ith bit of x), and Bob has an n-bit binary number $y = y_1y_2 \dots y_n$. Alice and Bob want to determine whether $x = y$. Computation is cheap, but communication is expensive; thus, they want to come up with some kind of scheme for passing messages back and forth which helps them figure out if $x = y$, and they want to transmit the smallest number of bits that they can. Here is the scheme they settle on:

1. Alice randomly chooses a prime number p which is greater than n and less than n^2.
2. Alice computes the number $h = x \bmod p$.
3. Alice sends p and h to Bob. (this costs about $\log p + \log h = O(\log n)$ bits of communication.)
4. Bob computes the number $g = y \bmod p$.
5. If $g = h$, then Bob sends back a 1 to Alice ("success!"). Otherwise, Bob sends back a 0 to Alice.

This scheme has 2 really good aspects:

- The communication complexity is small. Logarithmic is much better than it sounds: if $n = 10^9$ (x is a billion bits), then the message h is only 30 bits.
- If it's actually true that $x = y$, this scheme will correctly identify this. There are no false negatives!

However, there is a chance that this scheme gives a false positive. Here's the problem: Alice and Bob got their numbers x and y from an adversary who wants to force as many false positives as they can. So what the adversary will do is this: it will let $x = 0$, and then pick a special number K and let $y = K$. What makes K special? The idea here is that the adversary will choose K to make $x \bmod p = y \bmod p$ for as many values of p as possible. Since all the p-values are prime, the only way to do this is to choose K to be a product of as many primes between n and n^2 as possible.

Concrete Example

For concreteness, think about the example we looked at in class, where $n = 10$ (so $x, y \leq 2^{10} - 1 = 1023$). There are 21 primes between 10 and 10^2; they are $11, 13, 17, 19, 23, 29, 31, \dots, 89, 97$. So if the adversary chooses $K = 11 \cdot 13 = 143$, then the algorithm gives a false positive whenever $p = 11$ or when $p = 13$. If Alice picks one of the 21 primes uniformly at random, this false positive occurs 2 out of every 21 times the algorithm is run, with probability $2/21$.

Could the adversary do any better than this? In order to push the false positive rate higher, the adversary would have to choose K to be a product of more than just 2 of the primes. The smallest this product could be is $11 \cdot 13 \cdot 17 = 2431$, which is larger than x or y could possibly be when $n = 10$ (it's larger than 1023). So there's no way the adversary can force a false positive rate higher than $2/21$.

Assignment

Your assignment is to experimentally check how high of a false positive rate the adversary can force. **I want to see a plot with n on the horizontal axis and the worst false positive rate the adversary can force.** This false positive rate should be computed in 2 separate ways:

1. Theoretically, as above: Directly compute the largest number of primes greater than n whose product is less than $2^n - 1$, and divide this by the number of primes between n and n^2. The adversary shouldn't be able to do better than this.

2. Empirically: compute K, the *product* of the primes described in the previous point. Implement the fingerprinting scheme with $x = 0$ and $y = K$, with many trials of randomly-selected primes between n and n^2. Track the empirical false positive rate (how often is $y \bmod p = 0$). As you increase the number of trials, the empirical false positive rate computed here should converge to the theoretical false positive rate from the previous point.

My conjecture is that the false positive rate will generally be less than $1/n$, but I haven't solved this problem completely myself so I'm open to being surprised! Based on the examples we saw in class and above, we already know to expect that when $n = 6$, the false positive rate will be $1/8$; when $n = 10$, the false positive rate will be $2/21$. Given that you can download tabulations of many many prime numbers (e.g., https://github.com/koorukuroo/Prime-Number-List/tree/master), I think it should be reasonable to complete this assignment for all n from 6 to 1000; although there may come a point where the empirical false positive rate may not look like it's converging to the theoretical false positive rate because the probability of collisions becomes so low.

Additional Assignment for CS 5080

Write a function which gives an upper bound on the false positive rate without running empirical experiments. Use this function to generate a plot of an upper bound on the false positive rate for n from 10 to 10^{100} (1 googol). I recommend using a log-log plot for this. The specifics of how you do this are up to you. Here is one possibility, similar to what I suggested in class. Think of the false positive rate as a fraction N/D, where N is the largest number of primes which can be multiplied together while being less than $2^n - 1$, and D is the number of primes between n and n^2. So for $n = 10$, we had $N = 2$ and $D = 21$. You're trying to compute an upper bound on this fraction, which means you're free to *overestimate* N and *underestimate* D.

One overestimate for N is the largest m for which it is true that $n^m < 2^n - 1$. I.e., in real life, K is going to be a product of a bunch of different primes greater than n; howver, if you just pretend that all the primes are exactly equal to n, you'll be sure to overestimate N.

Underestimating D can be tricky, since we don't know quite as much about the exact number of primes. But I'd recommend appealing to the prime number theorem which says that the number of primes less than n is about $\frac{n}{\ln n}$. For our purposes here we might not be able to do much better than just pretending this is a formula for the number of primes, and how you turn that "formula" into an underestimate is up to you!