Math 110 Homework 6 Solutions

February 19, 2015

1. (a) Describe the Solovay-Strassen primality test (Chapter 6.3) and explain why it works.

   (b) Use the test on $n = 804\,509$. Is n composite, prime, or inconclusive?

   Solution: The Solovay-Strassen primality testing algorithm works analogously to the Miller-Rabin algorithm - for a base a check a congruence condition that holds if n is prime and hope that a randomly chosen a has a good chance of catching a composite n. In this case, choose a to be a unit modulo n, and check the congruence

   $$a^{\frac{n-1}{2}} \equiv \left(\frac{a}{n}\right) \pmod{n}.$$

   If it holds, we suspect n is prime: we repeat as necessary until we are willing to say that n is probably prime.

   The congruence holds when n is prime because of Euler's criterion. When n is not prime, one has to analyze the chance a randomly chosen a will catch this fact. It turns out that at least half will. This is not done in Trappe and Washington and this level of detail is not the point of this question. If you want to see this analysis, you can read it at `http://planetmath.org/solovaystrassentest`.

   Note that this test is computationally efficient as the exponentiation can be done by repeated squaring and the Jacobi symbol can be computed using quadratic reciprocity efficiently.

   Now let $n = 804509$. Let's try $a = 2$. Computing that $2^{402254} \equiv 651195 \pmod{n}$ which is not $\pm 1 \pmod{n}$ shows that n is not prime.

2. (a) Describe the $(p-1)$ factoring algorithm (Chapter 6.4) and explain why it works. What must be true of the factors of n for this algorithm to succeed quickly?

   (b) By choosing a base a and testing some small values of B, use the algorithm to find a factor of 49349.

   Solution: To factor n, we will try computing $b \equiv a^{B!} \pmod{n}$ and then checking whether $b-1$ shares any common factors with n using the Euclidean algorithm. Suppose $p|n$. Then $a^{p-1} \equiv 1 \pmod{p}$, and hence $a^{B!} \equiv 1 \pmod{p}$ provided $p-1|B!$. If all of the prime factors of $p-1$ are small, they will appear in $B!$ for a relatively small value of B. In that case, $b \equiv 1 \pmod{p}$. If we also had $b \not\equiv 1 \pmod{n/p}$, which is likely unless n/p only had small factors too, then the greatest common divisor of $b-1$ and n would be p. This would give us a way to factor n.

   Note that this is a specialized factorization algorithm: it only works if $p-1$ has small factors. The primes used in cryptography should be chosen so they do not have this property.

   Now let $n = 49349$. We pick a base $a = 2$ and try $B = 10$: we compute $a^{B!} \equiv 2^{10!} \equiv 18423 \pmod{n}$. The greatest common divisor of 18422 and n is 61. Hence 61 is a factor. (The full factorization is $61 \cdot 809$).

3. (a) Describe the Quadratic Sieve factorization method (Chapter 6.4) and explain why it works.

   (b) Let $n = 4181$. Find a factor of n using the following:

$$\begin{aligned}
65^2 &\equiv 44 \pmod{4181}\\
66^2 &\equiv 175 \pmod{4181}\\
67^2 &\equiv 308 \pmod{4181}\\
145^2 &\equiv 120 \pmod{4181}\\
429^2 &\equiv 77 \pmod{4181}\\
497^2 &\equiv 330 \pmod{4181}\\
688^2 &\equiv 891 \pmod{4181}
\end{aligned}$$

Solution: The basis idea is that knowing four square roots of a number modulo n allows us to factor n by computing greatest common divisors. The quadratic sieve produces these square roots as follows. Pick a factor base $\{p_1, \dots, p_r\}$ consisting of small primes. Square random integers modulo n and try to factor the result using only primes in the factor base. This is easy to do provided the factor base is small, for example by computing the greatest common divisor of n and the product of the primes in the factor base. Produce a lot of these equations

$$\begin{aligned}
a_1^2 &\equiv p_1^{m_{1,1}} \dots p_r^{m_{1,r}} \pmod{n}\\
a_2^2 &\equiv p_1^{m_{2.1}} \dots p_r^{m_{2,r}} \pmod{n}\\
&\dots\\
a_s^2 &\equiv p_1^{m_{s,1}} \dots p_r^{m_{s,r}} \pmod{n}.
\end{aligned}$$

Then find a product of the a_j^2's such that each exponent of a p_i is even. This can be interpretted as doing linear algebra over $\mathbb{F}_2$ to find a dependence relation between the vectors

$$(m_{1,1}, \dots, m_{1,r}), \dots, (m_{s,1}, \dots, m_{s,r}) \in \mathbb{F}_2^r.$$

Let A be the product of the a_j's appearing in the relation.

At this stage, we see that $A^2 \equiv p_1^{2s_1} p_2^{2s_2} \dots p_r^{2s_r} \pmod{n}$ for some integers $s_1, \dots, s_r$. There is a good chance that $A \not\equiv \pm p_1^{s_1} \dots p_r^{s_r} \pmod{n}$ which means we have multiple square roots of $A^2 \pmod{n}$ and hence can compute a gcd to obtain a factor of n.

In the example given, we want to combine the equations to get a square on the right hand side. We'll do this by inspection, but this can also be done by computing the kernel of a linear transformation over $\mathbb{F}_2$. Note that $65^2 \equiv 2^2 \cdot 11 \pmod{n}$ and $66^2 \equiv 5^2 \cdot 7 \pmod{n}$ and $429^2 \equiv 7 \cdot 11 \pmod{n}$. Therefore

$$(65 \cdot 66 \cdot 429)^2 \equiv 2^2 \cdot 5^2 \cdot 7^2 \cdot 11^2 \pmod{n}.$$

This gives square roots of $65 \cdot 66 \cdot 429 \equiv 770 \pmod{n}$ and $2 \cdot 5 \cdot 7 \cdot 11 \equiv 770 \pmod{n}$. So this didn't help. Instead, note that $67^2 \equiv 2^2 \cdot 7 \cdot 11$. Using that instead of 429, we get

$$(65 \cdot 66 \cdot 67)^2 \equiv 2^4 \cdot 5^2 \cdot 7^2 \cdot 11^2 \pmod{n}.$$

This shows that $65 \cdot 66 \cdot 67 \equiv 3122 \pmod{4181}$ and $4 \cdot 5 \cdot 7 \cdot 11 \equiv 1540 \pmod{4181}$ have the same square, but $-3122 \not\equiv 1540 \pmod{4181}$. Then $\gcd(3122 - 1540, 4181) = 113$ is a prime factor. The other is 37.

4. (a) Describe the Pollard rho algorithm.

   (b) Using $x_0 = 1$ and $f(x) = x^2 + 1$, find a factor of $n = 403$. At each step i, you may compute just $\gcd(x_i - x_{i-1}, n)$ (instead of performing all computations $\gcd(x_i - x_j, n)$ with $j < i$).

Solution: In the Pollard ρ algorithm, we fix a polynomial $f(x)$, often x^2+1. We choose a starting x_0, and then compute $x_1 = f(x_0) \pmod{n}$, $x_2 = f(x_1) \pmod{n}$, etc. After computing x_n, we check whether $\gcd(x_n - x_i, n) \neq 1, n$ for $i < n$. If this does not happen, we will eventually repeat an x value, in which case we have failed to factor n and need to retry with a different starting value and/or different polynomial.

The idea behind this algorithm is that iterating the polynomial f is like a random function. Doing it multiple times will eventually get back to place where you already where. You can think of it as defining a random walk on $\mathbb{Z}/n\mathbb{Z}$. Using the Chinese remainder theorem, we can equivalently do this random walk on each of the prime factors separately. Hopefully, one of the primes will return to a place it already was before the others do, so the gcd calculation will give a factorization of n. A more mathematical analysis is in Koblitz's book, Section V.2.

In the numerical example, we iterate and see that $x_0 \equiv 1 \pmod{403}$, $x_1 \equiv 2 \pmod{403}$, $x_2 \equiv 5 \pmod{403}$, $x_3 \equiv 26 \pmod{403}$ and $x_4 \equiv 274 \pmod{403}$. All of the $\gcd(x_i, x_{i-1}) = 1$ until we get that $\gcd(274 - 26, 403) = 31$. This shows that $403 = 31 \cdot 13$.