Number Theory Notes

MAT 331, Spring 2020

March 10, 2020

Definition 0.1. For integers a, b, n, we write $a \equiv b \bmod n$ if n divides $a - b$.

(The Python/Sage command `a%n` computes the unique integer b between 0 and $n - 1$ such that $a \equiv b \bmod n$).

Definition 0.2. If a, n are integers, an inverse of a modulo n is an integer b such that $ab \equiv 1 \bmod n$.

Theorem 0.1. *Let a, b be integers. Then a has an inverse modulo n if and only if a, n are relatively prime (i.e. share no common divisors).*

Some examples:

- 2 is invertible modulo 9. In fact $2 * 5 \equiv 1 \bmod 9$, so 5 is the inverse.
- 6 is *not* invertible modulo 9. This is because $6, 9$ share a common factor, 3, so they are not relatively prime.

In the case that a, n are relatively prime, an inverse can be found using the Extended Euclidean Algorithm (implemented in Sage as `xgcd`). Recall that if we apply this algorithm to (a, n), it returns integers d, e, f such that $ae+nf = d$, where $d = \gcd(a, n)$. Since a, n are relatively prime, $\gcd(a, n) = 1$. Hence we get that $ae + nf = 1$. Reducing this equation modulo n gives $ae \equiv 1 \bmod n$, which means that e is the inverse of a modulo n.

Theorem 0.2 (Fermat's little theorem)**.** *Let p be prime, and a be an integer relatively prime to p (since p is prime, this just means that p does not divide a). Then*

$$a^{p-1} \equiv 1 \bmod n.$$

Example: $3^{16} \equiv 1 \bmod 17$.

This can be generalized to the case when p is not prime. We define the Euler totient function $\phi(n)$ to be the number of positive integers less than n that are relatively prime to n. This is implemented in Sage as `euler_phi`.

Theorem 0.3 (Euler)**.** *Let a, n be relatively prime positive integers. Then*

$$a^{\phi(n)} \equiv 1 \bmod n.$$

Example: The positive integers less than 10 that are relatively prime to 10 are $1, 3, 7, 9$, so $\phi(10) = 4$. Taking $a = 3$, $n = 10$, we get that $3^4 \equiv 1 \bmod 10$.

Theorem 0.4 (Prime Number Theorem). *Let $\pi(N)$ denote the number of primes less than N. Then*

$$\pi(N) \sim N/\log N$$

This notation means that $\lim_{N\to\infty} \frac{\pi(N)}{N/\log N} \to \infty$.

The above result can be interpreted as follows: a randomly chosen integer near n has probability $1/\log n$ of being prime. For us, the relevance of this is that prime numbers are fairly common, since $\log n$ does not grow very quickly.

Primality Testing: Consider the problem of determining whether a given integer of n digits is prime. There are sophisticated algorithms that solve this problem in time $p(n)$, where p is a polynomial. The Sage function `is_prime` is such an algorithm. (The "trial-division" algorithm that you likely implemented in a previous homework takes time exponential in n).

On the other hand, factoring is conjectured to be harder (on "classical computers"; however if a sophisticated enough *quantum* computer could be built, factoring could be achieved much more efficiently. There have recently been importance advances towards building such a machine).

Assumption/Conjecture 0.1. *There is no algorithm that will factor an n digit integer into primes that runs in time polynomial in n.*

Modular exponentiation: The "repeated squaring" function `exp_mod` we wrote in class (equivalent to the Sage builtin function `power_mod`) computes a^e `%` b in time that is a polynomial in $\log(a), \log(e), \log(b)$.

On the other hand, doing the opposite, i.e. taking discrete logarithms, is conjectured to be hard.

Assumption/Conjecture 0.2. *There is no algorithm that, given a, b, c, will compute e such that $c = a^e$ `%` b (assuming such an e exists) and that runs in time polynomial in* $\log(a)\log(b), \log(c)$.

Overview of RSA To generate an RSA public key, private key pair, Alice chooses two large primes p, q (for instance by testing random large numbers using a fast primality testing algorithm, such as Sage's `is_prime`) and an integer b (we used $b = 17$) that is relatively prime to $(p-1)(q-1)$. She also computes $n = pq$. Her public key is (b, n), which she publishes to the world. To generate her private key, she computes e such that $be \equiv 1 \bmod (p-1)(q-1)$ (using the Extended Euclidean Algorithm, implemented in Sage as `xgcd`). This e is her private key.

If Bob wants to send Alice a message m (which we assume is an integer less than n), he computes the encrypted message as

$$m^b \bmod n,$$

which can be done efficiently using the repeated squaring trick (using Sage's `power_mod`).

If Alice receives an encrypted message c, she decrypts it by computing

$$c^e \bmod n,$$

using `power_mod`.