Computer Science 52

Some Facts from Number Theory

Fall Semester, 2014

These notes are adapted from a document that was prepared for a different course several years ago. They may be helpful as a summary of the facts that we use in Computer Science 52 as we study RSA encryption. The exercises in this document are left over from the previous course; although they are a tremendous amount of fun, they are not part of the assigned work for Computer Science 52.

1 Quotients and Remainders

If m and n are integers, and $n \neq 0$, then there is a unique pair of integers q and r satisfying

$$m = qn + r, \qquad \text{and} \qquad 0 \leq r < |n|.$$

Not surprisingly, q is the *quotient* and r is the *remainder* when m is divided by n.[1] We say that n *divides* m, or n is a *divisor* of m, when the remainder is zero. The notation $n \mid m$ is used to signify that n divides m.

An integer p is a *prime number* if $1 < |p|$ and the only divisors of p are ± 1 and $\pm p$.

We write $a \equiv b \pmod{n}$ if $b - a$ is divisible by n.[2] This notion, called congruence, is an equivalence relation. All the usual laws of arithmetic hold for congruence; for example, we have the distributivity of multiplication over addition and the associativity of multiplication.

$$a(b + c) \equiv ab + ac \pmod{n}$$
$$a(bc) \equiv (ab)c \pmod{n}$$

When computing arithmetic results for a congruence, it does not hurt replace a value by its remainder on division by n. This keeps the results in the range[3] between 0 and $|n|$.

[1] Be aware that different processors compute remainders differently. The remainder operator % in C may yield a result r which does not satisfy the stated inequality, but this will happen *only* when one or both of the arguments is negative.

[2] The notation is potentially confusing. Here, "mod" is not used as an operator; we are not saying that a is equivalent to (b mod n). Rather, the equivalence between a and b depends on n. A better notation might be $a \equiv_n b$.

[3] For encryption, n is most likely much larger than the largest possible integer. For this, we need multi-word data structures and special library routines for the arithmetic operations.

2 Divisors

Suppose that a and b are integers, with one or both being non-zero. An integer d is a *greatest common divisor* of a and b if

1. $d \mid a$,
2. $d \mid b$, and
3. if $e \mid a$ and $e \mid b$, then $e \mid d$.

According to this definition, if d is a greatest common divisor of a and b, then so is $-d$. We use the notation $\gcd(a,b)$ to denote the unique *positive* greatest common divisor. The integers a and b are *relatively prime* if $\gcd(a,b) = 1$.

The following theorem characterizes the greatest common divisor. As we shall see, it provides an extremely useful tool.

Theorem 1 *Suppose a and b are integers, at least one of which is non-zero, and let d be the least positive integer of the form $ua + vb$. Then $d = \gcd(a,b)$.*

PROOF: First observe that there are positive integers of the form $ua + vb$. (If a is positive, just take u to be 1 and v to be 0. If a is negative, take u to be -1 and v to be 0. If a is zero, then b is non-zero—take u to be 0 and v to be ± 1.) Therefore, there is a *least* positive integer d of the appropriate form.

Suppose that $d = u_0 a + v_0 b$. Divide a by d to obtain a quotient q and remainder r: $a = qd + r$ and $0 \le r < d$. Then $r = a - qd = (1 - qu_0)a + (-qv_0)b$, so r is of the form $ua + vb$. But d was the least positive value of that form, and r is less than d. Therefore r cannot be positive. The only possibility is for r to be zero, and in that case, d divides a. In the same way, we conclude that d divides b.

If e divides both a and b, then e divides the combination $u_0 a + v_0 b$. Hence e divides d. This completes the proof that $d = \gcd(a,b)$.

Corollary 2 *If $\gcd(a,n) = 1$, then there is an integer u such that $ua \equiv 1 \pmod{n}$.*

PROOF: There are integers u and v such that $ua + vn = 1$. We see that $ua - 1 = (-v)n$, so n divides $ua - 1$ and $ua \equiv 1 \pmod{n}$.

Corollary 3 *If $\gcd(a,n) = \gcd(b,n) = 1$, then $\gcd(ab,n) = 1$.*

PROOF: Again, there are integers u and v such that $ua + vn = 1$. Similarly, there are u' and v' such that $u'b + v'n = 1$. Multiply the left hand sides of the two equations to get another expression that equals 1:

$$(uu')ab + (uav' + u'bv + vv')n = 1.$$

The least positive combination of ab and n is therefore 1, and $\gcd(ab, n) = 1$.

Another useful idea is Euler's *totient* function, which is defined as follows. Let n be a non-zero integer. Then $\varphi(n)$ is the number of integers between 1 and $n-1$ (inclusive) which are relatively prime to n. That is, $\varphi(n)$ is the cardinality of the set $\{j \mid 1 \leq j < n \text{ and } \gcd(j, n) = 1\}$.

Exercise 2.1. Show, directly from the definition, that $\gcd(a, b)$ is unique.

Exercise 2.2. Suppose that a and n are relatively prime and n divides ab. Show that n divides b.

Exercise 2.3. Prove that a is relatively prime to b if and only if there are integers u and v such that $ua + vb = 1$.

Exercise 2.4. Prove that p is a prime number if and only if $\varphi(p) = p - 1$.

Exercise 2.5. If p and q are different primes, show that $\varphi(pq) = (p-1)(q-1)$.

Exercise 2.6. If p is a prime number and k is positive, what is the value of $\varphi(p^k)$?

---

3 Euclid's Algorithm

Theorem 1 is essential for characterizing the greatest common divisor, but it does not directly give a very efficient algorithm for computing the gcd. The following properties of the gcd function lead to Euclid's algorithm for computing the greatest common divisor.

Theorem 4 *For integers a and b, not both zero, we have the following properties:*

1. $\gcd(a, 0) = |a|$.
2. $\gcd(a, b) = \gcd(b, a)$.
3. $\gcd(a, b) = \gcd(-a, b)$.
4. *If $a = qb + r$, then* $\gcd(a, b) = \gcd(r, b)$.

PROOF: The first three properties can be proved directly from the definition of the greatest common divisor. For the fourth, note that

$$ua + vb = ur + (uq + v)b,$$

so that any number of the form $ua + vb$ can also be written in the form $u'r + v'b$. Conversely, any number of the latter form can be written as a combination of a and

```
a = abs(a0);  b = abs(b0);

while ((a != 0) && (b != 0))
      if (a < b)
         b = b % a;
      else
         a = a % b;

gcd = (a == 0) ? b : a;
```

Figure 1: Euclid's algorithm for computing the greatest common divisor.

---

b. The set of positive integers of the form $ua + vb$ is the same as the set of positive integers of the form $u'r + v'b$. The least element of the set is the same in both cases, and $\gcd(a, b) = \gcd(r, b)$.

Euclid's algorithm uses the properties of Theorem 4 to preserve loop invariants. Suppose that the we wish to compute the greatest common divisor of two integers whose values are stored in the C variables `a0` and `b0`. The C code below starts by establishing three invariants:

$0 \le$ `a`
$0 \le$ `b`
$\gcd($`a`$,$ `b`$) = \gcd($`a0`$,$ `b0`$)$.

This is easily done with the two assignments at the top of the code in Figure 1. Properties 2 and 3 from Theorem 4 show that the invariant is established.

The invariants are preserved by the loop of Figure 1. Even though `a` or `b` may be changed by an iteration of the loop, both variables remain non-negative. Moreover, property 4 from Theorem 4 guarantees that the gcd does not change.

On each iteration, either `a` or `b` is made smaller. One of the variables must eventually reach zero, and the loop will terminate. When it does, the other variable contains the greatest common divisor by property 1. This is the value selected by the last line in Figure 1.

For encryption, we often want to express the greatest common divisor in terms of the original numbers `a0` and `b0`,

```
gcd = u*a0 + v*b0.
```

In fact, the coefficients `u` and `v` are often more important than the value of the gcd itself.

It is easy to modify the Euclid's algorithm to provide the extra information. We simply use four new variables and maintain two additional invariants.

$$\begin{aligned} \mathsf{a} &= \mathsf{ua} * \mathsf{a0} + \mathsf{va} * \mathsf{b0} \\ \mathsf{b} &= \mathsf{ub} * \mathsf{a0} + \mathsf{vb} * \mathsf{b0} \end{aligned}$$

If we initialize the four variables correctly and preserve the invariants inside the loop, then one pair of coefficients—either ua and va or else ub and vb—will be the desired coefficients at the completion of the loop.

Exercise 3.1. Fill in the details in extending Euclid's algorithm to find u and v such that u*a0 + v*b0 is the greatest common divisor of a0 and b0.

Exercise 3.2. Show that Euclid's algorithm makes at most $2(\log_2 \mathsf{a0} + \log_2 \mathsf{b0})$ loop iterations. (In two iterations, either a or b becomes at least one bit shorter. Can you find a better bound?)

---

4 Fermat's Theorem

The following theorem is the heart of the RSA algorithm.

Theorem 5 (Fermat's Theorem) *If $n \neq 0$ and $\gcd(m, n) = 1$, then*

$$m^{\varphi(n)} \equiv 1 \pmod{n}.$$

PROOF: Consider the integers between 1 and $|n| - 1$ which are relatively prime to n. There are $\varphi(n)$ of them, and we can list them:

$$a_1,\ a_2,\ \ldots\ a_{\varphi(n)}. \tag{1}$$

For each i satisfying $1 \leq i \leq \varphi(n)$, let a'_i be the remainder upon division of ma_i by n. Both m and a_i are relatively prime to n, so by Corollary 3 the product ma_i is relatively prime to n. Further, by property 4 of Theorem 4, the remainder a'_i is relatively prime to n. Therefore, a'_i occurs among the elements of the list (1).

We next show that the elements in the list

$$a'_1,\ a'_2,\ \ldots\ a'_{\varphi(n)} \tag{2}$$

are all different. If $a'_j = a'_k$, then $ma_j \equiv ma_k \pmod{n}$. We have that n divides $m(a_j - a_k)$, so by the result of Exercise 2, n divides $a_j - a_k$. Because of the range of values of the a_i's, this means that $a_j = a_k$, or equivalently, that $j = k$. Therefore, the lists (1) and (2) contain exactly the same numbers, although perhaps in different orders.

Multiplying the numbers in either list gives the same result:

$$a_1 a_2 \ldots a_{\varphi(n)} = a'_1 a'_2 \ldots a'_{\varphi(n)},$$

or

$$\begin{aligned} a_1 a_2 \ldots a_{\varphi(n)} &\equiv m a_1 \, m a_2 \, \ldots \, m a_{\varphi(n)} \pmod{n} \\ &\equiv m^{\varphi(n)} \, a_1 a_2 \ldots a_{\varphi(n)} \pmod{n}. \end{aligned} \tag{3}$$

Each term in the product $a_1 a_2 \ldots a_{\varphi(n)}$ is relatively prime to n. Using Corollary 3 inductively, the whole product is relatively prime to n. By Corollary 2, there is a number u such that $u a_1 a_2 \ldots a_{\varphi(n)} \equiv 1 \pmod{n}$. Multiplication by u effectively "cancels" $a_1 a_2 \ldots a_{\varphi(n)}$ from the members of Equation (3), leaving the desired result $1 \equiv m^{\varphi(n)} \pmod{n}$.

Corollary 6 (Fermat's Little Theorem) *If p is a prime number, then for all integers m, $m \equiv m^p \pmod{p}$.*

Exercise 4.1. Prove Corollary 6, Fermat's Little Theorem. (Hint: Recall that $\varphi(p) = p - 1$.)

Exercise 4.2. The specific fact upon which the RSA algorithm rests concerns the product of two primes. If n is the product of two distinct primes p and q, prove that $m \equiv m^{1+\varphi(n)} \pmod{n}$, for all integers m. (The proof of this fact is sketched in the RSA paper.)

---

5 Industrial Strength Primes

Suppose that we have a number p that we think is prime. If we choose a random positive number a less than p, we can compute $a^p \bmod p$. If the result is different from a, then by Corollary 6, Fermat's Little Theorem, we know that p is *not* prime. If the result is a, then p may, or may not, be prime.

Now suppose that we repeat the test for many different values of a. If the two values are ever unequal, then we can declare with certainty that$p$ is not prime. If, after several tests, we do not discover evidence that p is non-prime, it is tempting to declare p to be a prime number. However, there is a chance that we will make a mistake and declare a number to be prime when it is in fact not. But by making many such tests, we can make the probability of such an error very low. A number declared prime on the basis of such a test is called an *industrial strength prime.*

Nearly all non-prime numbers will, with high probability, be exposed by this process, but there are a few exceptions. Non-primes that will always be declared industrial

strength primes by our method are called *Carmichael numbers.* Although there is an infinite number of them, they are rare—meaning that they are distributed sparsely among the integers. The smallest Carmichael number is 561. Using more sophisticated number theory, people have devised similar tests that do not have such a defect. These tests satisfy the two properties below:

- If p is prime, then every test will report it as prime.
- If p is not prime, then a given test will report it as prime with probability less than $1/2$. Therefore, if we make k tests, the probability of an error is at most $1/2^k$, a very small number.

Observe that we are not making a statement about "the probability that p is prime." A number is either prime or it is not; there is no chance involved. We are saying that, if a number is not prime, there is a small possibility that the test will give wrong information.

To find an industrial strength prime number, we simply generate a random number and test it, as above, several times. If it survives, we have our result. If not, we try again with a different random number.

Exercise 5.1. A question for thought: The strategy here is "guess a number and see if it is prime; if not, guess again." This seems as bad, or worse, than a brute-force search for primes. Why do we find primes after only a "reasonable" number of guesses?

---

6 Polynomial-time Algorithms

These notes have been evolving since 1996. At that time, industrial strength primality testing was the only technology available. More recently, in 2002, the AKS test for primality was developed. Refinements of it produce a deterministic test that tells us whether or not a k-bit integer is prime in $O(k^6)$ time. There is, however, no known way of factoring composite numbers in deterministic polynomial time.