Lecture Notes 9

1 Modular Arithmetic

There are several closely-related notions associated with "mod".

First of all, $\bmod$ is a binary operator. If $a \geq 0$ and $b \geq 1$ are integers, then $a \bmod b$ is the remainder of a divided by b. When either a or b is negative, there is no consensus on the definition of $\bmod$. We are only interested in $\bmod$ for positive b, and we find it convenient in that case to define $(a \bmod b)$ to be the smallest non-negative integer r such that $a = bq + r$ for some integer q. Under this definition, we always have that $r = (a \bmod b) \in \mathbf{Z}_b$. For example $(-5 \bmod 3) = 1 \in \mathbf{Z}_3$ since for $q = -2$, we have $-5 = 3 \cdot (-2) + 1$. Note that in the C programming language, the mod operator % is defined differently, so $a \,\%\, b \neq a \bmod b$ when a is negative and b positive.[1]

Mod is also used to define a relationship on integers:

$$a \equiv b \pmod{n} \quad \text{iff} \quad n \,|\, a - b.$$

That is, a and b have the same remainder when divided by n. An immediate consequence of this definition is that

$$a \equiv b \pmod{n} \quad \text{iff} \quad (a \bmod n) = (b \bmod n).$$

Thus, the two notions of mod aren't so different after all!

When n is fixed, the resulting two-place relationship $\equiv$ is an equivalence relation. Its equivalence classes are called *residue* classes modulo n and are denoted using the square-bracket notation $[b] = \{a \mid a \equiv b \pmod{n}\}$. For example, for $n = 7$, we have $[10] = \{\ldots -11, -4, 3, 10, 17, \ldots\}$. Clearly, $[a] = [b]$ iff $a \equiv b \pmod{n}$. Thus, $[-11]$, $[-4]$, $[3]$, $[10]$, $[17]$ are all names for the same equivalence class. We choose the unique integer in the class that is also in $\mathbf{Z}_n$ to be the *canonical* or preferred name for the class. Thus, the canonical name for the class containing 10 is $[10 \bmod 7] = [3]$.

The relation $\equiv \pmod{n}$ is a *congruence* relation with respect to addition, subtraction, and multiplication of integers. This means that for each of these arithmetic operations $\odot$, if $a \equiv a' \pmod{n}$ and $b \equiv b' \pmod{n}$, then $a \odot b \equiv a' \odot b' \pmod{n}$. This implies that the class containing the result of $a + b$, $a - b$, or $a \times b$ depends only on the classes to which a and b belong and not the particular representatives chosen. Hence, we can define new addition, subtraction, and multiplication as operations on equivalence classes, or alternatively, regard them as operations directly on $\mathbf{Z}_n$ defined by

$$\begin{aligned} a \oplus b &= (a + b) \bmod n \\ a \ominus b &= (a - b) \bmod n \\ a \otimes b &= (a \times b) \bmod n \end{aligned} \tag{1}$$

We remark that $\otimes$ is defined on all of $\mathbf{Z}_n$, but if a and b are both in $\mathbf{Z}_n^*$, then $a \otimes b$ is also in $\mathbf{Z}_n^*$.

[1] For those of you who are interested, the C standard defines $a \,\%\, b$ to be the number satisfying the equation $(a/b) * b + (a \,\%\, b) = a$. C also defines a/b to be the result of rounding the real number a/b towards zero, so $-5/3 = -1$. Hence, $-5 \,\%\, 3 = -5 - (-5/3) * 3 = -5 + 3 = -2$.

2 Modular Exponentiation and Euler's Theorem

Recall the RSA encryption and decryption functions

$$\begin{aligned} E_e(m) &= m^e \bmod n \\ D_d(c) &= c^d \bmod n \end{aligned}$$

where $n = pq$ is the product of two distinct large primes p and q. We see that both are based on modular exponentiation of large integers, an operation that we now explore in some depth.

We mentioned in lecture notes 8, section 2.3, that $\mathbf{Z}_n^*$ is an Abelian group under $\otimes$. This means that it satisfies the following properties:

Associativity $\otimes$ is an associative binary operation on $\mathbf{Z}_n^*$. In particular, $\mathbf{Z}_n^*$ is closed under $\otimes$.

Identity 1 is an identity element for $\otimes$ in $\mathbf{Z}_n^*$, that is $1 \cdot x = x \cdot 1 = x$ for all $x \in \mathbf{Z}_n^*$.

Inverses For all $x \in \mathbf{Z}_n^*$, there exists another element $x^{-1} \in \mathbf{Z}_n^*$ such that $x \cdot x^{-1} = x^{-1} \cdot x = 1$.

Commutativity $\otimes$ is commutative. (This is only true for *Abelian* groups.)

Example: Let $n = 26 = 2 \cdot 13$. Then

$$\mathbf{Z}_{26}^* = \{1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25\}$$

and

$$\phi(26) = |\mathbf{Z}_{26}^*| = 12.$$

The inverses of the elements in $\mathbf{Z}_{26}^*$ are given in Table 1. The bottom row of the table gives equiva-

Table 1: Table of inverses in $\mathbf{Z}_{26}^*$.

x	1	3	5	7	9	11	15	17	19	21	23	25
x^{-1}	1	9	21	15	3	19	7	23	11	5	17	25
$=$	1	9	-5	-11	3	-7	7	-3	11	5	-9	-1

lent integers in the range $[-12, \ldots, 13]$. This makes it apparent that $(26 - x)^{-1} = -x^{-1}$. In other words, the last row reads back to front the same as it does from front to back except that all of the signs flip from $+$ to $-$ or $-$ to $+$, so once the inverses for the first six numbers are known, the rest of the table is easily filled in.

It is not obvious from what I have said so far that inverses always exist for members of $\mathbf{Z}_n^*$,, and even showing that $\mathbf{Z}_n^*$ is closed under $\otimes$ takes a bit of work. Nevertheless, both are true. The latter isn't too hard for you to work out for yourself, and the former will become apparent later when we show how to compute the inverse.

Recall Euler's ϕ function which was defined in lecture notes 8, section 2.3 to be $|\mathbf{Z}_n^*|$, the cardinality of $\mathbf{Z}_n^*$. From the properties given there, one can derive an explicit formula for $\phi(n)$.

Theorem 1 *Write n in factored form, so $n = p_1^{e_1} \cdots p_k^{e_k}$. where $p_1, \ldots, p_k$ are distinct primes and $e_1, \ldots, e_k$ are positive integers.*[2] *Then*

$$\phi(n) = (p_1 - 1) \cdot p_1^{e_1 - 1} \cdots (p_k - 1) \cdot p_k^{e_k - 1}.$$

[2] By the fundamental theorem of arithmetic, every integer can be written uniquely in this way up to the ordering of the factors.

When p is prime, we have simply $\phi(p) = (p-1)$, and for the product of two distinct primes, $\phi(pq) = (p-1)(q-1)$. Thus, $\phi(26) = (13-1)(2-1) = 12$, as we have seen.

A general property of finite groups is that if any element x is repeatedly multiplied by itself, the result is eventually 1. That is, 1 appears in the sequence $x, (x \otimes x), (x \otimes x \otimes x), \ldots$, after which the sequence repeats. For example, for $x = 5$ in $\mathbf{Z}_{26}^*$, we get the sequence $5, 25, 21, 1, 5, 25, 21, 1, \ldots$. The result of multiplying x by itself k times can be written x^k. The smallest integer k for which $x^k = 1$ is called the *order* of x, sometimes written $\mathrm{ord}(x)$. It follows from general properties of groups that the order of any element of a group divides the order of the group. For $\mathbf{Z}_n^*$, we therefore have $\mathrm{ord}(x) \,|\, \phi(n)$. From this fact, we immediately get

Theorem 2 (Euler's theorem) $x^{\phi(n)} \equiv 1 \pmod{n}$ *for all* $x \in \mathbf{Z}_n^*$.

As a special case, we have

Theorem 3 (Fermat's theorem) $x^{(p-1)} \equiv 1 \pmod{p}$ *for all* x, $1 \le x \le p-1$, *where* p *is prime.*

Corollary 4 *Let* $r \equiv s \pmod{\phi(n)}$. *Then* $a^r \equiv a^s \pmod{n}$ *for all* $a \in \mathbf{Z}_n^*$.

Proof: If $r \equiv s \pmod{\phi(n)}$, then $r = s + u\phi(n)$ for some integer u. Then using Euler's theorem, we have

$$a^r \equiv a^{s+u\phi(n)} \equiv a^s \cdot (a^u)^{\phi(n)} \equiv a^s \cdot 1 \equiv a^s \pmod{n},$$

as desired. ■

The importance of this corollary to RSA is that it gives us a condition on e and d that ensures the resulting cryptosystem works. That is, if we require that

$$ed \equiv 1 \pmod{\phi(n)}, \tag{2}$$

then it follows from Corollary 4 that $D_d(E_e(m)) = m^{ed} \equiv m \pmod{n}$ for all messages $m \in \mathbf{Z}_n^*$, so $D_d()$ really does decrypt messages in $\mathbf{Z}_n^*$ that are encrypted by $E_e()$.

What about the case of messages $m \in \mathbf{Z}_n - \mathbf{Z}_n^*$? There are several answers to this question.

1. For such m, either $p \mid m$ or $q \mid m$ (but not both because $m < pq$). If Alice ever sends such a message and Eve is astute enough to compute $\gcd(m, n)$ (which she can easily do), then Eve will succeed in breaking the cryptosystem. So Alice doesn't really want to send such messages if she can avoid it.

2. If Alice sends random messages, her probability of choosing a message not in $\mathbf{Z}_n^*$ is only about $2/\sqrt{n}$. This is because the number of "bad" messages is only $n - \phi(n) = pq - (p-1)(q-1) = p + q - 1$ out of a total of $n = pq$ messages altogether. If p and q are both 512 bits long, then the probability of choosing a bad message is only about $2 \cdot 2^{512}/2^{1024} = 1/2^{511}$. Such a small probability event will likely never occur during the lifetime of the universe.

3. For the purists out there, RSA does in fact work for all $m \in \mathbf{Z}_n$, even though Euler's theorem fails for $m \notin \mathbf{Z}_n^*$. For example, if $m = 0$, it is clear that $(0^e)^d \equiv 0 \pmod{n}$, yet Euler's theorem fails since $0^{\phi(n)} \not\equiv 1 \pmod{n}$. We omit the proof of this curiosity.

3 Computation with Big Integers

The security of RSA depends on n, p, q being sufficiently large. What is sufficiently large? That's hard to say, but p and q are typically chosen to be roughly 512 bits long when written in binary, in which case n is about 1024 bits long. Already this presents a major computational problem since the arithmetic built into typical computers can handle only 32 bit integers (or 64 bit integers for the most advanced technology). This means that all arithmetic on large integers must be performed by software routines.

The straightforward algorithms for addition and multiplication that you learned in grade school have time complexities $O(N)$ and $O(N^2)$, respectively, where N is the length (in bits) of the integers involved. Asymptotically faster multiplication algorithms are known, but they involve large constant factor overheads, so it's not clear whether they are practical for numbers of the size we are talking about. What is clear is that a lot of cleverness is possible in the careful implementation of even the $O(N^2)$ multiplication algorithms, and a good implementation can be many times faster in practice than a poor one. They are also not particularly easy to implement correctly since there are many special cases that must be handled correctly.

Most people choose to use big number libraries written by others rather than write their own code. Two such libraries that you can use in this course are ln3 (the third in a succession of Large Number packages by René Peralta) and gmp (Gnu Multiprecision Package). Ln3 provides a nice C++ user interface but has some limitations on the size numbers that it can handle. I have made it available on the Zoo in `/c/cs467/ln3`. Documentation is in `/c/cs467/ln3/doc`. Gmp provides a large number of highly-optimized function calls for use with C and C++. It is preinstalled on all of the Zoo nodes and supported by the open source community. Type `info gmp` at a shell for documentation.

4 Exponentiation: Controlling Growth of Intermediate Results

We now turn to the basic operation of RSA, modular exponentiation of big numbers. This is the problem of computing $m^e \bmod n$ for big numbers m, e, and n.

The obvious way to compute this would be to compute $t = m^e$ and then compute $t \bmod n$. The problem with this approach is that m^e is too big! m and e are both numbers about 1024 bits long, so their values are each about 2^{1024}. The value of t is then $(2^{1024})^{2^{1024}}$. This number, when written in binary, is about $1024 * 2^{1024}$ bits long, a number far larger than the number of atoms in the universe (which is estimated to be only around $10^{80} \approx 2^{266}$). The trick to get around this problem is to do all arithmetic in $\mathbf{Z}_n$ using the equations (1), that is, reduce the result modulo n after each arithmetic operation. The product of two length ℓ numbers is only length 2ℓ before reduction mod n, so one never has to deal with numbers longer than about 2048 bits.