OWO Lecture: Modular Arithmetic with Algorithmic Applications

Martin Otto

Winter Term 2008/09

Contents

1 Basic ingredients

For the purpose of this stand-alone short lecture we want to take the following ingredients for granted:

– ordinary integer arithmetic with addition and multiplication.

– the notion of divisibility and of division with remainder.

– the notion of greatest common divisor.

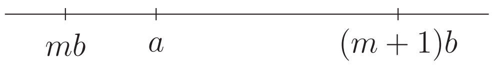
Here is a brief summary of these preliminaries.

Integer arithmetic. $\mathbb{Z}$ is the sets of integers, $\mathbb{N} \subseteq \mathbb{Z}$ the set of natural numbers. We regard 0 as a natural number. The basic arithmetical operations are addition and multiplication (and we tacitly assume the existence of feasible algorithms to perform integer addition and multiplication in terms of binary representations of the arguments and results).[1]

Divisibility. Let $a \in \mathbb{Z}$, $b \in \mathbb{N}$. Then b *divides* a if $a = mb$ for some $m \in \mathbb{Z}$. We write $b|a$ for "b divides a" or "a is divisible by b" or "b is a divisor of a".

A natural number $p > 1$ is a *prime* if it is divisible only by 1 and itself.

Division with remainder. Let $a, b \in \mathbb{Z}$, $b > 0$. The numbers mb for $m \in \mathbb{Z}$ are referred to as the (integer) multiples of b. For every $a \in \mathbb{Z}$ there is a unique $m \in \mathbb{Z}$ such that $mb \leqslant a < (m+1)b$.



In this case we refer to the value m, which is the best approximation from below of the quotient a/b in $\mathbb{Z}$, as $\lfloor a/b \rfloor$. The difference between a and mb is called the *remainder* of a w.r.t. [division by] b and denoted

$$a \bmod b,$$

[1]Feasible here means that the result is computed within a number of steps that is polynomial (here in fact linear and quadratic, respectively) in the lengths of the binary representations of the arguments (their *bit-size*). Note that there is an exponential relationship between the bit-size and the value of a number, as length k binary strings represent numbers up to $2^k - 1$. Seen from the other side, the bit-size is only logarithmic in the numerical value.

also read as "a modulo b". This remainder can equivalently be characterised as the unique $r \in \{0, \ldots, b-1\}$ for which $a - r$ is divisible by b (check this as an exercise!). We shall repeatedly use variants of the equation

$$a \bmod b = a - b\lfloor a/b \rfloor. \tag{1}$$

Greatest common divisors. Let $a, b > 0$. Then the *greatest common divisor* of a and b is the number

$$\gcd(a, b) := \max\{d \in \mathbb{N} \colon d|a \text{ and } d|b\}.$$

This definition is extended to the (degenerate) cases where one or both of a and b are zero, by putting $\gcd(a, 0) = \gcd(0, a) = a$ for all $a \in \mathbb{N}$.

Numbers a and b are called *relatively prime* if $\gcd(a, b) = 1$.

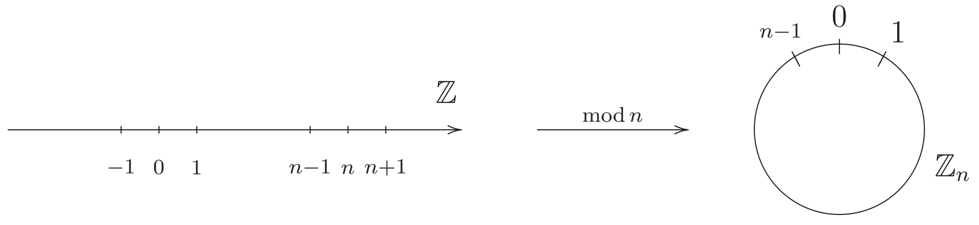
We state a fundamental fact about the gcd from elementary number theory. Part of this follows from the analysis of the extended form of Euclid's algorithm below, the other part will be an exercise.

Fact 1.1 *Let $a, b > 0$. Then the greatest common divisor $\gcd(a, b)$ is the smallest positive member of the set $\{ka + \ell b \colon k, \ell \in \mathbb{Z}\}$. This set precisely consists of all the integer multiples of $\gcd(a, b)$.*

2 Modular arithmetic

2.1 Going in circles

Arithmetic modulo n is the integer arithmetic of remainders w.r.t. n. We fix $n > 0$ and instead of the integers a we just look at their remainders $a \bmod n \in \{0, \ldots, n-1\} =: \mathbb{Z}_n$. This passage may be visualised as wrapping the integer line around a length n circle:



Formally this intention may be captured with notion of an *equivalence relation* (compare section 1.2.3 of the lecture notes), which lumps together all integers that leave the same remainder. Two integers a and b get lumped together if

$$a \bmod n = b \bmod n, \tag{2}$$

which is the case if, and only if, n divides their difference. This is the same as to say that $a = b + kn$ for some suitable $k \in \mathbb{Z}$, or that a and b are "equal up to an integer multiple of n". This motivates the following variant notation for equation (2)

$$a = b \pmod{n}, \tag{3}$$

which stresses the idea of "equality up to multiples of n".

Integer arithmetic in $\mathbb{Z}$ can be exported to the domain $\mathbb{Z}_n$ of remainders to obtain addition and multiplication operations in $\mathbb{Z}_n$ (see section 1.3 in the lecture notes), $+_n$ and $\cdot_n$. Here we are only interested in their compatibility with ordinary integer arithmetic

Lemma 2.1 *For all $a, b \in \mathbb{Z}$:*

$$\begin{aligned}(a+b) \bmod n &= \big((a \bmod n) + (b \bmod n)\big) \bmod n,\\ (a \cdot b) \bmod n &= \big((a \bmod n) \cdot (b \bmod n)\big) \bmod n.\end{aligned}$$

This means in particular that for arithmetical results modulo n, we may as well always already reduce the arguments and intermediate results modulo n, in order to minimise computational complexity.

Proof. We sketch the argument in the case of addition.

$a = kn + (a \bmod n)$ and $b = \ell n + (b \bmod n)$ for suitable $k, \ell \in \mathbb{Z}$ (in fact $k = \lfloor a/n \rfloor$ and $\ell = \lfloor b/n \rfloor$, but we don't care). Then

$$a + b = (k + \ell)n + (a \bmod n) + (b \bmod n)$$

implies the desired "equality modulo n".

□

Looking at exponentiation, which is iterated multiplication, we get the following corollary for modular exponentiation.

Corollary 2.2 *For all $a \in \mathbb{Z}$, $b \in \mathbb{N}$:*

$$a^b \bmod n = (\underbrace{a \cdot \cdots \cdot a}_{b\ times}) \bmod n = (a \bmod n)^b \bmod n,$$

where the intermediate results may be reduced modulo n after every multiplication step.

2.2 Feasible modular exponentiation

Addition and multiplication modulo n are computationally reduced to corresponding algorithms for integers and the computation of remainders, for which the usual feasible algorithms can be used.

Even modular exponentiation, $(a, b) \mapsto a^b \bmod n$ is feasibly computable. Note that iterated multiplication does not work, as b-fold iteration is exponential in the bit-size of b (length of its binary representation).

Representing the exponent b in binary, as

$$\langle b \rangle_2 = b_k \dots b_0, \qquad \text{so that } b = b_0 + b_1 2 + b_2 2^2 + \dots + b_k 2^k,$$

we have

$$a^b = a^{(b_0 + b_1 2 + b_2 2^2 + \dots + b_k 2^k)} = a^{b_0} a^{2 b_1} \cdot \dots \cdot a^{2^k b_k}.$$

So only certain 2^i-th powers of a need to be collected, namely those for which $b_i \neq 0$. The 2^i-th powers

$$a, a^2, a^4, a^8, \dots, a^{2^i}, a^{2^{i+1}} = \left(a^{2^i}\right)^2, \dots$$

can be generated by successive squaring. This yields the following feasible modular exponentiation procedure based on *repeated squaring.*

The variable c successively gets set to a, a^2, a^4, …, a^{2^k} (mod n) in lines 3 and 5, and the corresponding factors a^{2^i} are multiplied into d for those b_i that are not 0.

Modular-Exponentiation(a, b, n) $\langle b \rangle_2 = b_k \dots b_0$ in binary

```
    ⟨b⟩_2 = b_k ... b_0 in binary
1   d := 1
2   c := a
3   FOR $i = 0, \dots, k$ DO
4           IF $b_i = 1$ THEN $d := dc \bmod n$
5           $c := c^2 \bmod n$
6           OD
7   return $d$
```

Exercise 1 Show that $3^{444} + 4^{333}$ is divisible by 5, and that $2^{999} + 5^{999}$ as well as $5^{222} - 2^{222}$ are divisible by 7 (without ever so much as thinking of 3-digit decimal numbers for intermediate results).

2.3 Euclid's algorithm

The following observation holds the key to a feasible evaluation of the greatest common divisor of two (positive) integers.

Lemma 2.3 *For integers $a, b > 0$:* $\gcd(a, b) = \gcd(b, a \bmod b)$.

Proof. We can even show that for arbitrary $d \in \mathbb{N}$:

$$\big(\, d|a \text{ and } d|b \,\big) \quad \text{iff}^{\,2} \quad \big(\, d|(a \bmod b) \text{ and } d|b \,\big).$$

Equality of the gcds is then immediate.

Assume first that $d|a$ and $d|b$. We need to show that d then also divides $a \bmod b$. This follows from $a \bmod b = a - b\lfloor a/b \rfloor$, see equation (1) above.

Conversely, assume that $d|(a \bmod b)$ and $d|b$. We need to show that d then also divides a. Again this follows from equation (1), as $a = a \bmod b + b\lfloor a/b \rfloor$.
□

Lemma 2.3 supports the following procedure for computing the gcd. This algorithm, which was known to (but possibly not invented by) Euclid in the 4th century BC, is one of the most fundamental examples of a mathematical algorithm.

The basic form of Euclid's algorithm computes $\gcd(a, b)$ by a simple recursion which terminates within a number of iterations that is linear in the bit-size of the inputs (the length of a binary representation of the numbers a and b).

```
EUCLID($a, b$)                                  assume $a \geqslant b$

1    IF $b = 0$  THEN return $a$
2                ELSE return EUCLID($b, a \bmod b$)
```

Correctness is shown on the basis of Lemma 2.3 above. That the recursive call is executed only a logarithmic number of times (in terms of the numerical values) follows directly from the observation that two rounds through the loop at least halve both arguments.

[2] "iff" is shorthand for "if, and only if"

Exercise 2 Show that for $a_0 > b_0 > 0$,

$$\left.\begin{array}{rcl} (a_1, b_1) & = & (b_0, a_0 \bmod b_0) \\ \text{and } (a_2, b_2) & = & (b_1, a_1 \bmod b_1) \end{array}\right\} \quad \Rightarrow \quad a_2 < a_0/2 \text{ and } b_2 < b_0/2.$$

Hint: show that, whenever $a > b$, then $a \bmod b < a/2$.

An extended form of Euclid's algorithm produces not only the value of $\gcd(a, b)$ but also values for k and ℓ such that $\gcd(a, b) = ka + \ell b$ (compare Fact 1.1 above). This extra information is useful for solving simple equations in modular arithmetic and plays a key role in some cryptographic applications below.

```
Extended-Euclid($a, b$)                                        assume $a \geqslant b$
1    IF $b = 0$  THEN return $(a, 1, 0)$
2                ELSE    DO
3                        $(d, x, y) :=$ Extended-Euclid$(b, a \bmod b)$
4                        return $(d, y, x - \lfloor a/b \rfloor y)$
5                        OD
```

Extended-Euclid has the same recursive structure as Euclid, and performs just like Euclid w.r.t. to variable (and output) d.

That the output (d, k, ℓ) of Extended-Euclid(a, b) satisfies the equation

$$d = \gcd(a, b) = ka + \ell b. \tag{4}$$

is shown as follows.[3]

Equation (4) is clearly true in case of termination in line 1 (no recursive call).

We now show that every one additional recursive call preserves the desired equation. We assume that line 3 returns d, x, y satisfying the requirement w.r.t. Extended-Euclid$(b, a \bmod b)$. Then it follows by simple arithmetic that the return value $(d, y, x - \lfloor a/b \rfloor y)$ satisfies the requirement for Extended-Euclid(a, b). Indeed,

$$d = \gcd(b, a \bmod b) = xb + y(a \bmod b)$$

[3]Formally, this is a proof by induction on the number of recursive calls to the Extended-Euclid procedure during the computation of Extended-Euclid(a, b). Compare section 1.2.6 of the lecture notes.

implies that $d = \gcd(a, b)$ (Lemma 2.3). It also implies that

$$ya + (x - \lfloor a/b \rfloor y)b = xb + y(a - \lfloor a/b \rfloor b) = xb + y(a \bmod b) = d$$

as required (compare equation (1)).

2.4 Solving modular equations

We want to find solutions to simple linear equations in modular arithmetic, like finding $x \in \mathbb{Z}_n = \{0, \dots, n-1\}$ such that

$$ax = b \pmod n, \tag{5}$$

for given n and $a, b \in \mathbb{Z}_n$.

Lemma 2.4 *The modular equation* (5) *is solvable for* x *iff* $\gcd(n, a) | b$.

Proof. Assume first that the equation has a solution $x = \ell$ say. So $b = \ell a + kn$ for some $k \in \mathbb{Z}$. But $\gcd(n, a)$ divides all numbers of the form $kn + \ell a$ by Fact 1.1.

Conversely, let $\gcd(n, a)$ divide b: $b = m \gcd(n, a)$ for some $m \in \mathbb{Z}$. From Fact 1.1 we know that $\gcd(n, a) = kn + \ell a$ for suitable $k, \ell \in \mathbb{Z}$.

So $b = mkn + m\ell a$, and $x := m\ell \bmod n$ solves the equation.

$\square$

We may assume that $0 \leqslant a, b < n$, i.e., that $a, b \in \mathbb{Z}_n$ (else pass to $a \bmod n$ and $b \bmod n$ without changing the meaning of the equation).

If $\gcd(n, a)|b$, then a solution to equation (5) is easy to find on the basis of Extended-Euclid as follows.

Extended-Euclid(n, a) yields (d, k, ℓ) such that $d = \gcd(n, a)$ and $d = kn + \ell a$. Therefore $d = \ell a \pmod n$. It follows that if $d|b$, then

$$x = (\ell b/d) \bmod n \quad \text{is a solution.}$$

This is because $d = \ell a \pmod n$ implies $a(\ell b/d) = \ell ab/d = db/d = b \pmod n$.

Example 2.5 Consider the modular equation $ax = 1 \pmod n$. It is solvable if and only if a and n are relatively prime. In this case an output $(1, k, \ell)$ from Extended-Euclid(n, a) yields the solution $x = \ell \bmod n$.

For $93x = 1 \pmod{100}$, for instance, Extended-Euclid$(100, 93)$ gives $d = 1$, $k = 40$, $\ell = -43$. The solution for $x \in \{0, \dots, 99\}$ is obtained as $-43 \bmod 100 = 57$.

3 Fermat's Little Theorem

The following theorem of Fermat plays an important role in primality tests as well as for the arithmetic of certain cryptographic protocols.

Theorem 3.1 (Fermat) *For all primes p and all $a \in \{1, \dots, p-1\}$:*

$$a^{p-1} = 1 \pmod{p}. \tag{6}$$

Proof. You can find elementary proofs in textbooks, but we here sketch a more playful combinatorial proof. It proceeds via counting bead patterns.

(a) We count the number $S(n, a)$ of colour patterns in strings of n beads made from beads of a distinct colours. Clearly there are exactly a^n such patterns, as each position in the string allows us to choose one out of the a many colours. $S(n, a) = a^n$.

(b) Now let us count the number $C(n, a)$ of distinct circular colour patterns in bead cycles of length n. Look at a particular pattern. We may cut it open in n different places to form a string. For some circular patterns, different cuts will produce the same string pattern, namely if there is some internal periodicity within the string.

Let us say that a circular pattern has *period* q if a shift through q positions along the cycle does leave the pattern unchanged (every bead takes the place of a bead of the same colour). For instance, if two colours alternate, all even numbers are periods; if all beads are of the same colour then all numbers are periods. Zero and the overall length n will always be periods. Also, the smallest non-zero period must divide the length n (exercise!).

So if $n = p$ is a prime, then the smallest period can only be 1 or p. If it is 1, the pattern consists of just one colour; otherwise it is p and every cutting point yields a distinct string pattern.

(c) Now let $n = p$ a prime and $1 \leqslant a < p$. Then $K := C(p, a) - a$ is the number of circular patterns that use more than one colour. These K patterns give rise to pK distinct string patterns when we cut them open in all possible positions. This number also is $S(p, a) - a = a^p - a$. So we find

$$pK = a^p - a.$$

As a divides the right-hand side, a must divide the left-hand side and therefore K: p is a prime, so unless $a = 1$, a does not divide p. So $K = ak$ for some $k \in \mathbb{Z}$, and $kp = a^{p-1} - 1$. Hence $a^{p-1} = 1 + kp$ and equation (6) follows.

$\square$

4 Applications in cryptography

For a simple cryptographic setting, consider the problem of transmitting a message M from B (Bob) to A (Alice) over some insecure channel in such a way that even though someone might intercept the message, its contents is not revealed to the eavesdropper. To achieve this, B has to *encrypt* the message and only submit its encrypted form to the insecure channel; the necessary assumptions being that (a) a potential eavesdropper who intercepts the encrypted message cannot (or at least not easily) retrieve the original message; but that (b) the legitimate recipient, A, can *decrypt* the encrypted message to get at the original message.

Encryption and decryption are based on transformations of M which require specific extra information, so-called *keys*, to be (feasibly) computed. Of course, encryption and decryption are performed by means of matching pairs of transformations, such that the decryption transformation inverts the encryption transformation.

With simple symmetric schemes the same key is necessary for encryption and matching decryption (compare section 1.1.4 in the lecture notes), which raises the problem of making the key used by B (for encryption) available to A (for decryption). This *key exchange problem* can be overcome in surprising manners. We briefly look at two now classical mechanisms.

4.1 Interactive key exchange

The gist of the key exchange problem seems to be that in order to communicate securely, Alice and Bob must already share an exclusive secret (their key) beforehand. Can this constraint be avoided?

Such methods rest on the assumption that certain functions are easy to compute but hard to invert, so-called *one-way functions*. An important example is provided by modular exponentiation (see above). Though not a proven fact, there is good reason to assume that finding solutions to exponential modular equations

$$g^x = y \pmod n$$

for x is computationally hard (known as the discrete logarithm problem).

Modular exponentiation is used in a key exchange protocol idea due to Diffie, Hellman and Merkle (1976) as follows.

Alice and Bob agree (publicly) on suitable numbers n and g. In a session in which they want to generate a common key for cryptographic purposes, Alice and Bob independently choose numbers a and b. They do not reveal the numbers a and b to each other, however. Instead Alice sends to Bob the number $g^a \bmod n$ and Bob sends to Alice the number $g^b \bmod n$. According to the crucial complexity assumption, a and b are not accessible even to someone who intercepts these messages. Now Alice chooses for her key the number $(g^b \bmod n)^a \bmod n = g^{ab} \bmod n$, which she can compute from the number received from Bob using her own secret a. Bob computes the same number according to $g^{ab} \bmod n = (g^a \bmod n)^b \bmod n$ from the number received from Alice using his secret b.

Now Alice and Bob share the key $g^{ab} \bmod n$, without having communicated any information from which this key could be extracted (if the discrete logarithm is indeed not feasibly computable).

4.2 Public key systems

Public key cryptosystems like RSA overcome the key exchange problem by letting each participant have two keys: one public to be used for encryption by anyone who wants to communicate to the owner of this public key; and one secret known only to its owner which is necessary to decrypt any message thus encrypted.

So each participant is responsible for his or her own pair of matching keys, only one of which is made available to the rest of the world. The theoretical problem to be solved, once this idea is formulated, concerns a mechanism according to which such key pairs can be generated such that one key cannot feasibly be computed from the other (while at the same time, they can be used to feasibly compute the encryption and decryption transformations). Rivest, Shamir and Adleman (1977) devised the following scheme.

RSA is also based on modular exponentiation. We consider simple one-way communication from B to A. With a pair of two distinct large primes p and q (arbitrarily chosen by A) associate $n = pq$ and $N := (p-1)(q-1)$. Assume that n is sufficiently large and that the message is (encoded as) a natural number $M < n$.

A also chooses some integer e relatively prime to N, i.e., one such that $\gcd(N, e) = 1$. A then computes a solution d to the modular equation

$$ex = 1 \pmod{N}.$$

A's keys are the pairs of numbers $P = (n, e)$ (the public key) and $S = (n, d)$ (the secret key). The associated encryption and decryption transformations for messages $M, M' \in \{0, \dots, n-1\}$ are:

Encryption based on the *public key* $P = (n, e)$: $\quad M \longmapsto M^e \bmod n.$

Decryption based on the *secret key* $S = (n, d)$: $\quad M' \longmapsto (M')^d \bmod n.$

We show that these transformations provide a matching encryption/decryption pair:

Claim 4.1 *For all $M \in \{0, \dots, n-1\}$:* $\quad (M^e)^d = M \pmod{n}.$

Proof. The proof uses Fermat's Little Theorem (Theorem 3.1).
Since $ed = 1 \pmod{N}$ for $N = (p-1)(q-1)$, we have

$$ed = 1 + k(p-1)(q-1)$$

for suitable $k \in \mathbb{Z}$. Therefore,

$$M^{ed} = MM^{(p-1)(q-1)k} \pmod{n}.$$

Fermat's little theorem, applied for each of the primes p and q separately, implies that

$$M^{ed} = M \pmod{p} \quad \text{and} \quad M^{ed} = M \pmod{q}.$$

Thus $M^{ed} = M + k_1 p = M + k_2 q$ for suitable $k_i \in \mathbb{Z}$. Now $k_1 p = k_2 q$ for distinct primes p and q implies that $p | k_2$ and $q | k_1$. Using for instance that $k_2 = k_3 p$ for some $k_3 \in \mathbb{Z}$ we find that

$$M^{ed} = M + k_2 q = M + k_3 pq = M + k_3 n = M \pmod{n}.$$

□

It is important to note that the arithmetic in key generation, encryption and decryption is feasibly computable. We have indicated the crucial ingredients above. Another essential is the availability of large primes p and q to start with. For this one has randomised algorithms which feasibly generate (certified) primes. For security, RSA obviously relies on hardness of the discrete logarithm and also on the assumption that factorisation of large numbers into their prime factors is hard, for otherwise one could retrieve p and q and hence all the key information from the public key.