2WF15 - Discrete Mathematics 2 - Part 1

Algorithmic Number Theory

Benne de Weger

version 0.54, March 6, 2012

Contents

Chapter 1

Multi-precision arithmetic

Introduction

General references for this chapter: [CP, Chapter 9], [GG, Chapters 2, 8], [Kn, Chapter 4], [Sh, Chapter 3].

In this chapter the following topics will be treated:

- representation of integers and polynomials in a computer,
- efficient algorithms for elementary arithmetic operations for the following objects:
  - integers,
  - polynomials.

1.1 Notation

First the necessary notation. We use the following notation for sets of numbers.

- $\mathbb{N} = \{1, 2, 3, \ldots\}$ is the set of *natural numbers* (excluding 0),
- $\mathbb{Z} = \{\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots\}$ is the set of *integral numbers* or *integers*,
- $\mathbb{Q}$ is the set of *rational numbers*,
- $\mathbb{R}$ is the set of *real numbers*.

If a set S contains 0, we sometimes write $S^* = S \setminus \{0\}$.

The cardinality of a set S is denoted by $|S|$, or by $\#S$.

We will use the *floor* function $\lfloor x \rfloor$, which is the largest integer that is at most x, and the *ceiling* function $\lceil x \rceil$, which is the smallest integer that is at least x.

The symbol log will always stand for the natural logarithm. The logarithm to the base b will be denoted by $\log_b$.

To get an idea of the growth of $f(x)$ for $x \to \infty$, we say that a function $f(x)$ is of the *order of* $g(x)$ for some useful function g, if there exists a constant c such that $f(x) < cg(x)$ for all large enough x. The notation for this is the *big O*-notation:

$$f(x) = \mathrm{O}(g(x)).$$

For $g(x)$ we usually take simple functions such as x^α, $e^{\alpha x}$, $\log x$, etc. Of course there is a similar big O concept for e.g. $x \downarrow 0$.

Example: $\dfrac{n^2+3}{2\sqrt{n}-1000000\log n} = \mathrm{O}(n^{\frac{3}{2}})$. Or, if we want to be a bit more accurate: $\dfrac{n^2+3}{2\sqrt{n}-1000000\log n} = \dfrac{1}{2}n^{\frac{3}{2}} + \mathrm{O}(n\log n)$.

1.2 Representation of integers

A problem with doing arithmetic with computers is that numbers can only be stored in rather small memory words, and that computer arithmetic therefore can, at first sight, only work with small sized numbers. Most computers nowadays have a 32 bit architecture, some have 64 bit capabilities, but there are also many processors that still have 16 or 8 bit registers only, e.g. smartcard processors. In an n bit word we can represent 2^n different numbers, e.g. $\{0, 1, 2, \ldots, 2^n - 1\}$. In many applications, notably cryptography, we often need to do *exact* arithmetic with numbers that require hundreds or thousands of bits. In other words, overflow or rounding errors are not tolerated at all. This means that the standard arithmetic functions provided by the processor or the (operating system) software, operating on single (or at best double) words only, are usually not sufficient, and we must find efficient ways to do arithmetic with large integers.

When we want to store large integers that do not fit in the word size of some computer, we have to use an array of words to represent one integer. There are many different ways to represent integers. We don't want to get into too much implementation detail issues such as *big / little endian* or *2-complement* representations. Instead we'll keep it at a fairly mathematical level. We also do not treat how to represent (approximations of) non-integral rational or real numbers. Basically what we do is to use a so-called radix b representation, which is a generalization of the familiar decimal notation.

Let $b \geq 2$ be an integer, the so called *radix*. The elements of the set $\{0, 1, \ldots, b-1\}$ are called *radix b digits*, and we assume that they all fit in one memory word, and that operations on them can be handled by the available processor. Note that this means that the bit length of a word is $\geq \lceil \log_2 b \rceil$.

Which value we take as b will depend on the architecture of our computer. Popular choices are $b = 2^8, 2^{16}, 2^{32}$, reflecting the word size (2 to the power the bit size of a word), as this optimizes the storage. However, as we'll see soon, it may also be wise to take e.g. $b = 2^{31}$ or even $b = 2^{16}$ on a 32-bit computer.

Any integer $n \in \mathbb{Z}$ admits a radix b representation $[n]_b$, as follows. Let $m \in \mathbb{N}$ be such that $|n| < b^m$, so that $m \geq \lfloor \log_b |n| \rfloor + 1$. Then, repeatedly using division with remainder by b, it follows that there exist $n_0, n_1, \ldots, n_{m-1} \in \{0, 1, \ldots, b-1\}$ such that

$$|n| = n_0 + n_1 b + n_2 b^2 + \ldots + n_{m-1} b^{m-1} = \sum_{i=0}^{m-1} n_i b^i.$$

Then as the *radix b representation* of n we write

$$[n]_b = \pm [n_{m-1}, n_{m-2}, \ldots, n_0]_b,$$

with $\pm = +$ or left out if $n \geq 0$, and $\pm = -$ if $n < 0$. Leading zeroes are allowed, but usually are omitted, e.g.

$$[43]_{16} = [2, 11]_{16} = [0, 0, 2, 11]_{16}.$$

The parameter m is called the *word size* of $[n]_b$.

When it is clear from the context which b is meant, we often write $[n]_b = \pm n_{m-1}n_{m-2}\dots n_0$. Omitting commas should only be done when confusion is not possible. When $b \leq 10$ we use the well known arabic symbols $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$ for the digits. When $b > 10$ (but not too large) we run out of symbols, so after $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$ we sometimes continue with A, B, C, ….

When $b = 10$ we get essentially our every day *decimal representation*. When $b = 2$ we get the well known *binary representation*; in that case the word size is equal to the *bitsize*. When $b = 16$ the system is called *hexadecimal representation*, and in that case often the digits $10, 11, 12, 13, 14, 15$ are written as A, B, C, D, E, F. Example: $[43]_{16} = 2B$. This hexadecimal system is well known in the computing world.

1.3 Efficient arithmetic with integers

1.3.1 Addition and subtraction

Addition and subtraction is essentially easy: we use the primary school method, which is the most efficient way of doing it. This works by adding the digits backwards, and keeping track of "carries".

We start with a formal description of *addition* of positive numbers. Addition of a positive and a negative number is equivalent to subtraction of two positive numbers, to be treated below. Addition of two negative numbers is equivalent to addition of two positive numbers, and adjusting the sign of the output. Addition of 0 is trivial.

Algorithm 1.1 (Addition Algorithm)

Input:	an integer $b \geq 2$, and the radix b representations $[x]_b = [x_{m-1}, \dots, x_0]_b$, $[y]_b = [y_{n-1}, \dots, y_0]_b$ (without leading zeroes) of numbers $x, y \in \mathbb{N}$
Output:	the radix b representation $[z]_b = [z_{k-1}, \dots, z_0]_b$ (without leading zeroes) of $z = x + y$
Step 1:	$c \leftarrow 0$, $x_i \leftarrow 0$ for $m \leq i < \max\{m, n\}$ $y_i \leftarrow 0$ for $n \leq i < \max\{m, n\}$
Step 2:	for $i = 0, 1, \dots, \max\{m, n\} - 1$ do $\quad z_i \leftarrow x_i + y_i + c$ $\quad$ if $z_i \geq b$ then $z_i \leftarrow z_i - b$, $c \leftarrow 1$ else $c \leftarrow 0$
Step 3:	if $c = 1$ then $k \leftarrow \max\{m, n\} + 1$, $z_{k-1} = 1$ else $k \leftarrow \max\{m, n\}$ output $[z_{k-1}, \dots, z_0]_b$

Note that indeed $z_i \in \{0, 1, \dots, b-1\}$ for all i.

Next we give a formal description of *subtraction* of a positive number from a larger positive number. Subtraction of a positive number from a smaller positive number is essentially the same, by swapping the two numbers and adjusting the sign of the output. Subtraction of a positive and a negative number is equivalent to addition of two positive numbers, and adjusting the sign of the output. Subtraction of two negative numbers is equivalent to subtraction of two positive numbers and adjusting the sign of the output. Subtraction of 0 or from 0 is trivial.

Algorithm 1.2 (Subtraction Algorithm)

Input:	an integer $b \geq 2$, and the radix b representations $[x]_b = [x_{m-1}, \ldots, x_0]_b$, $[y]_b = [y_{n-1}, \ldots, y_0]_b$ (without leading zeroes) of numbers $x, y \in \mathbb{N}$ such that $x > y$ (so $m \geq n$)
Output:	the radix b representation $[z]_b = [z_{k-1}, \ldots, z_0]_b$ (without leading zeroes) of $z = x - y$
Step 1:	$c \leftarrow 0$, $y_i \leftarrow 0$ for $n \leq i < m$
Step 2:	for $i = 0, 1, \ldots, m-1$ do $\quad z_i \leftarrow x_i - y_i - c$ $\quad$ if $z_i < 0$ then $z_i \leftarrow z_i + b$, $c \leftarrow 1$ else $c \leftarrow 0$
Step 3:	$k \leftarrow m$ while $k \geq 2$ and $z_{k-1} = 0$ do $k \leftarrow k - 1$ output $[z_{k-1}, \ldots, z_0]_b$

Note that these addition and subtraction algorithms have linear complexity, i.e. the number of *atomic operations* (addition / subtraction operations on words) needed to add / subtract two numbers of radix b length n is $O(n)$ (in essence it is n, if we neglect operations on indices, on bits and on signs). Clearly one cannot do better. You should have noted that these algorithms may use variables that sometimes will be one bit longer than the word size. We do not go into details on how to deal with this (one possible solution is to use b one less than the bit size of a word).

1.3.2 Multiplication

1.3.2.1 Naive

Again *multiplication* can be done pretty efficiently by the naive, or primary school method. Given $x = \sum_{i=0}^{m-1} x_i b^i$ and $y = \sum_{j=0}^{n-1} y_j b^j$, we want to compute

$$xy = \left(\sum_{i=0}^{m-1} x_i b^i \right) \left(\sum_{j=0}^{n-1} y_j b^j \right) = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} x_i y_j b^{i+j}.$$

The method computes

$$\sum_{j=0}^{n-1} (xy_j) b^j \qquad \text{using} \qquad xy_j = \sum_{i=0}^{m-1} (x_i y_j) b^i \qquad \text{for} \qquad j = 0, 1, \ldots, n-1.$$

A multiplication operation on two words $x_i y_j$ yields a number that in general will not fit in one word anymore. Note that $0 \leq x_i y_j \leq (b-1)^2 < b^2$, so it does always fit in two words. The *carry* is the most significant word of the two. If to such a product a carry is to be added, say c of word length 1, we have $x_i y_j + c \leq (b-1)^2 + (b-1) = b^2 - b < b^2$, and so the new carry will also have word length 1. With induction it's now easy to prove that the carry will never be larger than one word.

This is implemented in the algorithm below, for the multiplication of two positieve numbers. When non-positive numbers have to be multiplied, you can use this algorithm too, and you simply have to adjust the sign.

Algorithm 1.3 (Naive Multiplication Algorithm)

Input:	the radix $b \geq 2$, and the radix-b-representations $[x]_b = [x_{m-1}, \ldots, x_0]_b$, $[y]_b = [y_{n-1}, \ldots, y_0]_b$ (without leading zeroes) of numbers $x, y \in \mathbb{N}$
Output:	the radix-b-representation $[z]_b = [z_{k-1}, \ldots, z_0]_b$ (without leading zeroes) of $z = xy$
Step 1:	$z_i \leftarrow 0$ for $n \leq i < m+n-1$
Step 2:	for $i = 0, 1, \ldots, m-1$ do $\quad c \leftarrow 0$ $\quad$ for $j = 0, 1, \ldots, n-1$ do $\quad\quad t \leftarrow z_{i+j} + x_i y_j + c$ $\quad\quad c \leftarrow \left\lfloor \frac{t}{b} \right\rfloor$ $\quad\quad z_{i+j} \leftarrow t - cb$ $\quad z_{i+n} \leftarrow c$
Step 3:	if $z_{m+n-1} = 0$ then $k \leftarrow m+n-2$ else $k \leftarrow m+n-1$ output $[z_{k-1}, \ldots, z_0]_b$

Many variants exist, that usually do not do it essentially better. Another idea is to mimic the formula

$$\left(\sum_{i=0}^{m-1} x_i b^i\right)\left(\sum_{j=0}^{n-1} y_j b^j\right) = \sum_{k=0}^{m+n-2}\left(\sum_{i=0}^{m-1} x_i y_{k-i}\right) b^k,$$

i.e. gather all products of digits that end up in the same digit of the product.

Note that to multiply two words (an atomic operation), the processor must be able to deal with double word length. One way to achieve this is to use only half the word size (i.e. have word size $> 2\log_2 b$). Standard processors have more efficient ways to do this. We do not go into details.

The complexity of this type of multiplication of two numbers of radix b lengths m and n respectively is $O(mn)$. If the complexity of addition and subtraction is neglected, it is essentially mn. Because often $m = n$ is taken, this is called *quadratic complexity* (as opposed to linear complexity for addition), because the complexity of multiplying two n bit numbers is $O(n^2)$.

1.3.2.2 Karatsuba

In the previous sections we noticed that multiplication by the primary school method has quadratic complexity: the number of atomic operations (additions or multiplications on words) you have to perform in order to multiply two n-word integers is $O(n^2)$. On the other hand, addition and subtraction have only linear complexity. That is a notable difference, and the question arises whether we can multiply more efficiently. Surprisingly, the answer is affirmative.

The speed of multiplication can be improved quite easily, both in theory and in practice. This is not obvious, and a lot of theory has been developed. The easiest way to show that improvement can be reached is by a trick known as *Karatsuba multiplication*. It is based on three nice ideas.

The first basic idea is to split up the numbers to be multiplied in halves, and perform only multiplications on the halves. So let us take two numbers x, y that have a radix b representation of even (bit) length n (if n is odd, add a leading zero). Then we can write

$$x = x_{\text{hi}} b^{n/2} + x_{\text{lo}}, \quad y = y_{\text{hi}} b^{n/2} + y_{\text{lo}},$$

where $x_{\text{hi}}, x_{\text{lo}}, y_{\text{hi}}, y_{\text{lo}}$ are numbers with a length $m/2$ representation. Then

$$xy = x_{\text{hi}}y_{\text{hi}}b^n + (x_{\text{hi}}y_{\text{lo}} + x_{\text{lo}}y_{\text{hi}})b^{n/2} + x_{\text{lo}}y_{\text{lo}}.$$

At first sight one would guess that this takes 4 multiplications of half-length numbers, and a (linear, therefore negligible) shift and add operation (note that multiplication by powers of b is just shifting, and essentially for free). Assuming that a multiplication of two numbers of size n costs cn^2 units (unit: atomic operation, or nanoseconds, or whatever reasonable measure you like), this naive method will cost $4 \times c(n/2)^2 + \mathrm{O}(n) = cn^2 + \mathrm{O}(n)$ units, and we have gained nothing.

The second basic idea now is the nice trick found by Karatsuba. Note that

$$(x_{\text{hi}} + x_{\text{lo}})(y_{\text{hi}} + y_{\text{lo}}) = x_{\text{hi}}y_{\text{hi}} + (x_{\text{hi}}y_{\text{lo}} + x_{\text{lo}}y_{\text{hi}}) + x_{\text{lo}}y_{\text{lo}}.$$

Now compute $x_{\text{hi}}y_{\text{hi}}$, $x_{\text{lo}}y_{\text{lo}}$, and $(x_{\text{hi}} + x_{\text{lo}})(y_{\text{hi}} + y_{\text{lo}})$, which is only 3 half-length multiplications and some additions. Then we can compute $x_{\text{hi}}y_{\text{lo}} + x_{\text{lo}}y_{\text{hi}} = (x_{\text{hi}} + x_{\text{lo}})(y_{\text{hi}} + y_{\text{lo}}) - x_{\text{hi}}y_{\text{hi}} - x_{\text{lo}}y_{\text{lo}}$ by doing only additions and subtractions, and to get xy we can compose the results by shifting and additions.

This method costs $3 \times c(n/2)^2 + \mathrm{O}(n) = \frac{3}{4}cn^2 + \mathrm{O}(n)$ units, and with this factor of $\frac{3}{4}$ we have found a faster method.

The third basic idea is to apply the trick recursively: also the half-length multiplications can be sped up in the same way. We will now see what this leads to in terms of complexity of the overall multiplication.

Assume that the resulting algorithm for multiplying two n-word numbers has complexity $T(n)$. Multiplying two numbers of word length $2n$ then costs 3 times $T(n)$, plus some linear terms. So there exists a $C > 0$ such that

$$T(2n) < 3T(n) + Cn.$$

If we now enlarge C, if necessary, to obtain $C > T(2)$, then using induction it easily follows that $T(2^k) < C(3^k - 2^k)$ for $k \geq 1$. Indeed,

$$T(2^k) < 3T(2^{k-1}) + C2^{k-1} < 3C(3^{k-1} - 2^{k-1}) + C2^{k-1} = C(3^k - 2^k).$$

With $k = \lceil \log_2 n \rceil$ we find

$$T(n) \leq T(2^k) < C \cdot 3^k < 3C \cdot 3^{\log_2 n} < 3C \cdot n^\alpha \quad \text{with} \quad \alpha = \frac{\log 3}{\log 2} = 1.5849\ldots \quad .$$

In other words: $T(n) = \mathrm{O}(n^\alpha) = \mathrm{O}(n^{1.585})$. And this is a substantial theoretical improvement over the naive quadratic algorithm.

In practice Karatsuba's method becomes advantageous already for rather small numbers (from about 5 words on).

1.3.2.3 Other methods

Karatsuba's idea can be generalized as follows. Split up x, y into $r + 1$ pieces of n words each:

$$x = x_r b^{rn} + x_{r-1}b^{(r-1)n} + \ldots + x_0, \quad y = y_r b^{rn} + y_{r-1}b^{(r-1)n} + \ldots + y_0.$$

Consider the polynomials

$$X(t) = x_r t^r + x_{r-1}t^{r-1} + \ldots + x_0, \quad Y(t) = y_r t^r + y_{r-1}t^{r-1} + \ldots + y_0,$$

and put

$$Z(t) = X(t)Y(t) = z_{2r}t^{2r} + z_{2r-1}t^{2r-1} + \ldots + z_0.$$

Since $x = X(b^n), y = Y(b^n)$ and $xy = Z(b^n)$, we are done when we can efficiently compute the $2r+1$ coefficients of Z. This we do by computing the values of $Z(t) = X(t)Y(t)$ at $t = 0, 1, 2, \ldots, 2r$, costing $2r + 1$ multiplications of n-word numbers, and then by interpolation the coefficients can be found in linear time, as they are fixed linear combinations of the computed numbers (the linear combinations are given by the inverse Vandermonde matrix on $0, 1, \ldots, 2r$). With $T(n)$ as above, we now find $T((r + 1)n) < (2r + 1)T(n) + Cn$. Taking C large enough so that $C > T(r + 1)$ we find by induction that $T\left((r+1)^k\right) \leq \frac{C}{r}\left((2r+1)^k - (r+1)^k\right)$. It follows, like above, that $T(n) < \frac{2r+1}{r} C n^{\log_{r+1}(2r+1)}$, and thus $T(n) < 3Cn^\alpha$ for $\alpha = \frac{\log(2r+1)}{\log(r+1)}$. With $r \to \infty$ we find:

$$\text{for every } \epsilon > 0 \ : \quad T(n) < cn^{1+\epsilon}.$$

This is quite spectacular, as it shows that multiplication can be done in almost (more precise: asymptotically) linear time. But this really is an asymptotic result, as c will depend on r, and in practice these generalized Karatsuba methods with $r > 1$ are not very useful. A better idea, due to Toom and Cook, is to let r vary with n, but we won't go into details here.

Completely different method exist, such as Fast Fourier Transforms, that can be practically used to achieve multiplication in essentially linear complexity. We do not go in details, see [Sh, Secton 18.6], [CP, Section 9.5], [GG, Section 8.2].

1.3.3 Division

Division with remainder is the problem of, given $x, y \in \mathbb{N}$ with $x > y$, computing the unique pair $q, r \in \mathbb{N}$ with $x = qy + r$ and $0 \leq r < y$. This is done by the primary school *long division* method. The words of the quotient q are computed from left to right. When a new word q_i has been found, the numerator x is decreased by $q_i y$. Then the above steps are repeated to find the next word q_{i-1}, until the numerator has become smaller than y. Then q is the number consisting of the words q_i, and r is the remaining numerator.

We give the following algorithm, in which you should notice that we did not write out all steps in atomic operations. So to implement this a lot of further work is needed. See [Kn, Section 4.3.1] or [Sh, Section 3.3.4] for more implementation details.

Algorithm 1.4 (Division with Remainder Algorithm)

Input:	the radix $b \geq 2$, and the radix b representations $[x]_b = [x_{m-1}, \ldots, x_0]_b$, $[y]_b = [y_{n-1}, \ldots, y_0]_b$ (without leading zeroes) of numbers $x, y \in \mathbb{N}$
Output:	the radix b representations $[q]_b = [q_{k-1}, \ldots, q_0]_b$ and $[r]_b = [r_{\ell-1}, \ldots, r_0]_b$ (without leading zeroes) of q, r such that $x = qy + r$ and $0 \leq r < y$
Step 1:	$r \leftarrow x$ $k \leftarrow m - n + 1$
Step 2:	for $i = k-1, k-2, \ldots, 0$ do $\quad q_i \leftarrow \left\lfloor \frac{r}{b^i y} \right\rfloor$ $\quad r \leftarrow r - q_i b^i y$
Step 3:	remove leading zeroes from $q = [q_{k-1}, \ldots, q_0]_b$ output $[q]_b$ and $[r]_b$

It remains to explain how to find the most significant word $\left\lfloor \frac{r}{b^i y} \right\rfloor$ of the quotient. This can be done by dividing the two most significant words of the numerator by the most significant word of the denominator. When the most significant word of the denominator is larger than $\frac{1}{2}b$, this is a good approximation (off by at most 2) of q_i. When the most significant word of the denominator is smaller than $\frac{1}{2}b$, simply multiply numerator and denominator by a suitable number to make the denominator's most significant word larger than $\frac{1}{2}b$.

So assume we have to compute the most significant word of $\frac{Ab^{m-1}+Bb^{m-2}+C}{Db^{m-2}+E}$, where $0 < A < b$, $0 \le B < b$, $0 \le C < b^{m-2}$, $\frac{1}{2}b \le D < b$, and $0 \le E < b^{m-2}$. Instead we compute the most significant word of $\frac{Ab+B}{D}$, which we regard as an almost elementary operation on single words only. Now note that

$$\begin{aligned}
\left| \frac{Ab^{m-1}+Bb^{m-2}+C}{Db^{m-2}+E} - \frac{Ab+B}{D} \right| &= \left| \frac{C}{Db^{m-2}+E} - \frac{(Ab+B)E}{(Db^{m-2}+E)D} \right| \le \\
&\le \max\left\{ \frac{C}{Db^{m-2}+E}, \frac{(Ab+B)E}{(Db^{m-2}+E)D} \right\} \le \\
&\le \max\left\{ \frac{b^{m-2}-1}{\frac{1}{2}b^{m-1}+0}, \frac{((b-1)b+b-1)(b^{m-2}-1)}{(\frac{1}{2}b^{m-1}+0)\frac{1}{2}b} \right\} < \\
&< \max\left\{ \frac{2}{b}, 4 \right\} \le 4,
\end{aligned}$$

and consequently the most significant words differ at most by 4 (with a more careful analysis one can actually do better and reach a difference of at most 2). So a small correction to the initially computed q and r may be necessary, and this can easily be arranged by doing $(q,r) \leftarrow (q,r) \pm (1,-y)$ at most 4 times, until $0 \le r < y$.

The complexity of division of two numbers of radix b lengths m and n is $\mathrm{O}(m(m-n+1))$.

1.4 Efficient polynomial arithmetic

We now make a few remarks on polynomial arithmetic.

1.4.1 Representation of polynomials

Let K be a field, often a finite one. In this section we assume that there is some efficient way of representing field elements, and to do the field operations.

When $K = \mathbb{Z}_p$ for a prime number p that fits in one computer word, this is easy. Then the atomic operations as described above can easily be adapted to this situation: one only has to take care of reducing $\mod p$ whenever necessary.

When $K = \mathbb{Z}_p$ for a larger prime, that does not fit in one word anymore, each field element will be represented by an array of words, and each field operation will become more involved. Efficient multi-precision arithmetic in $\mathbb{Z}_n$ will be treated in the next Chapter.

Anyway, such operations on field elements are now treated as "atomic operations". Thus note that an array of field elements may now have to be represented as a 2-dimensional array of computer words.

With this in mind, there is a lot of similarity between radix b representation:

$$[n]_b = [n_{m-1}, \ldots, n_0]_b \quad \text{representing} \quad n = \sum_{i=0}^{m-1} n_i b^i,$$

and representing polynomials by listing their coefficients:

$$[f] = [f_m, \ldots, f_0] \quad \text{representing} \quad f(X) = \sum_{i=0}^{m} f_i X^i.$$

1.4.2 Polynomial arithmetic

Addition, multiplication and division with remainder can now be treated in a similar way to the corresponding operations for integers. The only significant difference is that now there are no carries anymore. In fact, this makes polynomial arithmetic easier than integer arithmetic.

Also Karatsuba multiplication goes through for polynomials.

Exercises

1.1. Show that for given $a \in \mathbb{Z}$, $b \in \mathbb{N}$ there exists a unique pair of integers (q, r) such that $a = qb + r$ and $0 \leq r < b$.

1.2. Let $m \in \mathbb{N}$, and let $n \in \mathbb{Z}$ be such that $|n| < b^m$. Give a formal proof that there exist $n_0, n_1, \ldots, n_{m-1} \in \{0, 1, \ldots, b-1\}$ such that $|n| = \sum_{i=0}^{m-1} n_i b^i$.

1.3. Show that the radix b representation is unique when leading zeroes are not allowed.

1.4. Give an algorithm that, on input of radices b_1, b_2 and $[n]_{b_1}$, outputs $[n]_{b_2}$. Hint: start with $b_1 = 10, b_2 = 2$.

1.5. Compute the radix 2, 8 and 16 representations of 2181 and -3507.

1.6. When the radix b representation of n is given by $[n]_b = [n_{m-1}, n_{m-2}, \ldots, n_0]_b$, then what is for $k > 1$ the radix b^k representation $[n]_{b^k}$?

1.7. Work out the formulas for the cases $r = 1, 2$ of the generalized Karatsuba multiplication method.

1.8. Describe in detail efficient algorithms for polynomial addition, subtraction, multiplication, Karatsuba multiplication, and division with remainder.

1.9. Implement all algorithms given in this Chapter, and all you did in Exercise 1.8, in your favourite computer language or computer algebra system.

Chapter 2

Euclidean and Modular Algorithms

Introduction

General references for this chapter: [BS, Chapters 4, 5], [CP, Chapter 9], [GG, Chapters 3, 4, 5], [Sh, Chapters 4, 11]. And for those preferring Dutch: [Be, Chapters 3, 6], [Ke, Chapter 11], [dW, Chapters 1, 2].

In this Chapter the following topics will be treated:

- an analysis of the well known Euclidean Algorithm,
- algorithms for efficient multi-precision modular arithmetic,
- an algorithmic view of the Chinese Remainder Theorem,

2.1 The Euclidean Algorithm

2.1.1 The Greatest Common Divisor

The *greatest common divisor* of two integers a, b is the largest integer d such that $d \mid a$ and $d \mid b$. Notation: $\gcd(a, b)$ (or simply (a, b)). Note that $\gcd(a, 0) = |a|$.

When $\gcd(a, b) = 1$ we say that a and b are *coprime* or *relatively prime*.

Let $L(a, b) = \{xa + yb : x, y \in \mathbb{Z}\}$ be the set of $\mathbb{Z}$-*linear combinations* of a and b. The set $L(a, b)$ is closely linked to $\gcd(a, b)$, as is shown in the following lemma.

Lemma 2.1 *$L(a, b)$ is the set of multiples of* $\gcd(a, b)$.

Proof. Clearly, any common divisor of a and b is a divisor of any $xa + yb$, thus of each element of $L(a, b)$. So $L(a, b)$ contains only multiples of $\gcd(a, b)$. But we need to prove more: that it contains *all* multiples.

Let u be the smallest positive element of $L(a, b)$. Say that $u = xa + yb$. We now perform division with remainder on a and u. There exist q, r such that $a = qu + r$ and $0 \leq r < u$. As $r = a - qu = (1 - qx)a + (-qy)b$ we have $r \in L(a, b)$. But, by the definition of u, there are no

elements in $L(a,b)$ between 0 and u, so by $0 \leq r < u$ we must have $r = 0$. This implies $a = qu$, i.e. $u \mid a$. Similarly we prove that $u \mid b$, and it follows that $u \mid \gcd(a,b)$. But we already saw that $\gcd(a,b) \mid u$, so $\gcd(a,b) = u \in L(a,b)$, and hence also all multiples of $\gcd(a,b)$ are in $L(a,b)$. □

An important result is the following, expressing the gcd as a $\mathbb{Z}$-linear combination of its arguments.

Theorem 2.2 (gcd and linear combination) *For any $a, b \in \mathbb{Z}$ there exist $x, y \in \mathbb{Z}$ such that $\gcd(a,b) = xa + yb$.*

Proof. Immediate from Lemma 2.1. □

2.1.2 The basic Euclidean Algorithm

The treatment of the gcd in Section 2.1.1 was theoretical: we proved results about existence and properties of certain numbers, but that does not necessarily give a method to compute the gcd. The prime factorizations of a, b can be used to compute $\gcd(a,b)$, and this may be useful for small numbers, but finding the prime factorization of large numbers is a notoriously difficult problem in itself.

There is a very efficient way to compute $\gcd(a,b)$ without knowing the factorizations: the *Euclidean Algorithm*. It can also be used to check for coprimality.

Algorithm 2.1 (Euclidean Algorithm)

Input:	$a, b \in \mathbb{Z}$
Output:	$d \in \mathbb{Z}$ `such that` $d = \gcd(a,b)$
Step 1:	$a' \leftarrow \lvert a \rvert$, $b' \leftarrow \lvert b \rvert$
Step 2:	`while` $b' > 0$ `do`
	$\quad r \leftarrow a' - \left\lfloor \frac{a'}{b'} \right\rfloor b'$
	$\quad a' \leftarrow b'$, $b' \leftarrow r$
Step 3:	$d \leftarrow a'$, `output` d

Theorem 2.3 (Euclidean Algorithm) *The Euclidean Algorithm is correct, i.e. on input $a, b \in \mathbb{Z}$ it outputs $\gcd(a,b)$ in a finite number of steps.*

Proof. In step 1 we have $\gcd(a,b) = \gcd(a',b')$. Any common divisor of a' and b' is also a common divisor of $a' - qb'$ and b', for any $q \in \mathbb{Z}$, and vice versa. So when in the while-loop of Step 2 the number r is computed, the set of common divisors of a' and b' is the same as the set of common divisors of b' and r. So over the entire while-loop the set of common divisors of a' and b' is invariant, hence so is $\gcd(a',b')$. Further, in Step 2 the value of b' decreases strictly but remains nonnegative, as $0 \leq r < b'$. Eventually b' must become 0. At that point Step 2 ends, and $\gcd(a,b) = \gcd(a',b') = \gcd(a',0) = a'$. □

2.1.3 The Extended Euclidean Algorithm

The Euclidean Algorithm can be extended to output also the x and y of Theorem 2.2. This is commonly called the *Extended Euclidean Algorithm*.

Algorithm 2.2 (Extended Euclidean Algorithm)

Input:	$a, b \in \mathbb{Z}$
Output:	$d, x, y \in \mathbb{Z}$ `such that` $d = \gcd(a,b) = xa + yb$
Step 1:	$a' \leftarrow \lvert a \rvert$, $b' \leftarrow \lvert b \rvert$
	$x_1 \leftarrow 1$, $x_2 \leftarrow 0$
	$y_1 \leftarrow 0$, $y_2 \leftarrow 1$
Step 2:	`while` $b' > 0$ `do`
	$q \leftarrow \left\lfloor \frac{a'}{b'} \right\rfloor$, $r \leftarrow a' - qb'$
	$a' \leftarrow b'$, $b' \leftarrow r$
	$x_3 \leftarrow x_1 - qx_2$, $y_3 \leftarrow y_1 - qy_2$
	$x_1 \leftarrow x_2$, $y_1 \leftarrow y_2$
	$x_2 \leftarrow x_3$, $y_2 \leftarrow y_3$
Step 3:	$d \leftarrow a'$
	`if` $a \geq 0$ `then` $x \leftarrow x_1$ `else` $x \leftarrow -x_1$
	`if` $b \geq 0$ `then` $y \leftarrow y_1$ `else` $y \leftarrow -y_1$
	`output` d, x, y

Theorem 2.4 (Extended Euclidean Algorithm) *The Extended Euclidean Algorithm is correct, i.e. on input $a, b \in \mathbb{Z}$ it outputs $\gcd(a,b), x, y$ such that $\gcd(a,b) = xa + yb$ in a finite number of steps.*

Proof. After Theorem 2.3 we only have to prove that $\gcd(a,b) = xa + yb$. This follows at once from the invariancy in the while-loop of $a' = x_1|a| + y_1|b|$ and $b' = x_2|a| + y_2|b|$. □

2.1.4 A binary variant

The Euclidean Algorithm dates back to antiquity: Euclid lived around the year 300 BC. It is a so elegant algorithm that it should come as a surprise that it can be improved, as was only noticed recently (second half of the 20th century).

By far the most expensive operation in the (Extended) Euclidean Algorithm is the computation of q (long division). To avoid this computation, the *Binary Euclidean Algorithm* has been developed. It replaces division of a by b with repeated subtraction. This is usually faster, though the number of iterations may grow.

The underlying ideas are:

- if both a and b are even, then divide out a common factor 2; this multiplies the gcd by 2, and $\max\{a,b\}$ decreases;
- if exactly one of a and b is even, then divide the even number by 2; this does not change the gcd, and does not increase $\max\{a,b\}$;
- if both a and b are odd, then $a - b$ is even, so replace the larger one of a and b by $|a - b|$; this does not change the gcd, and $\max\{a,b\}$ decreases.

Algorithm 2.3 (Binary Euclidean Algorithm)

Input:	$a, b \in \mathbb{Z}$
Output:	$d \in \mathbb{Z}$ `such that` $d = \gcd(a, b)$
Step 1:	$a' \leftarrow \lvert a \rvert$, $b' \leftarrow \lvert b \rvert$
	$d \leftarrow 1$
	`while` a' `and` b' `are both even do` $a' \leftarrow \frac{1}{2}a'$, $b' \leftarrow \frac{1}{2}b'$, $d \leftarrow 2d$
	`while` a' `is even do` $a' \leftarrow \frac{1}{2}a'$
	`while` b' `is even do` $b' \leftarrow \frac{1}{2}b'$
	`if` $a' < b'$ `then` $(a', b') \leftarrow (b', a')$
Step 2:	`while` $b' > 0$ `do`
	$\quad a' \leftarrow a' - b'$
	$\quad$ `while` $a' > 0$ `and` a' `is even do` $a' \leftarrow \frac{1}{2}a'$
	$\quad$ `if` $a' < b'$ `then` $(a', b') \leftarrow (b', a')$
Step 3:	$d \leftarrow da'$, `output` d

We mention the following result without proof.

Theorem 2.5 (Binary Euclidean Algorithm) *The Binary Euclidean Algorithm is correct, i.e. on input $a, b \in \mathbb{Z}$ it returns $\gcd(a, b)$ in a finite number of steps. The number of times the while-loop is executed is at most $\mathrm{O}(\log \max\{a, b\})$.*

The bit complexity is now easily seen to be $\mathrm{O}((\log \max\{a, b\})^2)$, as in each while-loop only subtractions and shifts take place.

2.1.5 Complexity

2.1.5.1 Time

We now estimate the number of iterations in the (Extended) Euclidean Algorithm in more detail.

The first time the loop in Step 2 is executed, the value of q may be zero, namely when $a' < b'$. In that case Step 2 simply swaps a' and b'. But from then on always $b' < a'$, hence $q \geq 1$. Clearly the speed at which b' decreases is minimal if q is minimal, i.e. $q = 1$. This happens all the time exactly when a, b are consecutive Fibonacci numbers.

The *Fibonacci numbers* F_n (for $n = 0, 1, 2, \ldots$) are defined recursively as follows:

$$F_0 = 0, \quad F_1 = 1, \quad F_{n+1} = F_n + F_{n-1} \quad \text{for } n \in \mathbb{N}.$$

The sequence of Fibonacci numbers starts as

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, \ldots.$$

Because $F_{n-1} < F_n$ when $n \geq 3$, we find that $\dfrac{F_{n+1}}{F_n} = 1 + \dfrac{F_{n-1}}{F_n} < 2$, hence the integral part of $\dfrac{F_{n+1}}{F_n}$ is equal to 1, and the remainder is F_{n-1}.

It follows that when we apply the (Extended) Euclidean Algorithm to $a = F_{n+1}$ and $b = F_n$, the values of a', b' and q in the beginning of the while-loop are given by the sequence $(a', b', q) = (F_{n+1}, F_n, 1), (F_n, F_{n-1}, 1), (F_{n-1}, F_{n-2}, 1), \ldots, (3, 2, 1), (2, 1, 2)$, and finally $(a', b') = (1, 0)$. The while-loop is executed n times, and clearly this is the worst case.

We now estimate F_n. Let $\tau = \frac{1}{2}(1 + \sqrt{5}) = 1.618\ldots$ (the well known *golden section number*), and $\bar{\tau} = \frac{1}{2}(1 - \sqrt{5}) = -0.618\ldots$.

Lemma 2.6 *$F_n = \frac{1}{\sqrt{5}}(\tau^n - \bar{\tau}^n)$ for all $n = 0, 1, 2, \ldots$.*

Proof. Use mathematical induction. □

Corollary 2.7
(a) *F_n is the integer nearest to $\frac{1}{\sqrt{5}}\tau^n$ for all $n = 0, 1, 2, \ldots$.*
(b) *$n - 1 = \lceil \log_\tau F_n \rceil$ for $n = 3, 4, \ldots$.*

Proof. Trivial. □

We now have the following result, showing that the Euclidean Algorithm is indeed efficient: in the worst case the number of times the while-loop is executed is logarithmic in the input variables.

In practice this means that we can compute gcd's of pretty large integers very quickly: with e.g. 1000 digit integers the number of steps in the Euclidean Algorithm is at most 4800, which on a modern personal computer should be a matter of milliseconds.

Theorem 2.8 (Euclidean Algorithm, complexity) *When the (Extended) Euclidean Algorithm is applied to a, b with $b \geq 2$, then the number of times the while-loop is executed is at most $\lceil \log_\tau \min\{|a|, |b|\} \rceil$.*

Proof. Without loss of generality we may assume that $a > b > 0$. Let n be the integer such that $F_n \leq b < F_{n+1}$. Note that the result is easily verified for all cases with $b = 2$.
Hence $b \geq 3$, and $n \geq 4$. We show by induction that the number of iterations is at most $n - 1$. Then we are done, as by Corollary 2.7 $n - 1 = \lceil \log_\tau F_n \rceil \leq \lceil \log_\tau b \rceil$.
To get the induction started, note that the result is easily verified for all cases with $b = 3, 4$. So we now assume $b \geq 5$, hence $n \geq 5$.
In the first iteration of the while-loop we arrive at $a' = b$ and $b' = r = a - qb$.
If $b' \leq 2$ then the total number of iterations is at most $1 + 2 = 3 \leq n - 1$.
If $3 \leq b' \leq F_n$ then mathematical induction tells us that the total number of iterations is at most $1 + (n - 2) = n - 1$.
If $b' > F_n$ then we enter a second iteration. We denote the q, r of this second iteration by q^*, r^*. Note that $\frac{a'}{b'} = \frac{b}{b'} < \frac{F_{n+1}}{F_n} < 2$, hence in this second iteration we have $q^* = 1$, and $r^* = a' - b' = b - r < F_{n+1} - F_n = F_{n-1}$.
If $r^* \leq 2$ then the total number of iterations is at most $2 + 2 = 4 \leq n - 1$.
If $r^* \geq 3$ then mathematical induction tells us that the total number of iterations is at most $2 + (n - 3) = n - 1$. □

However, note that each step in the while-loop requires a (long) division. This implies that the so called *bit complexity* of the Euclidean Algorithm for n bit numbers is $O(n^3)$: the while-loop is executed $O(n)$ times, and each time a long division of quadratic complexity is done.

On looking a bit more careful, the bit complexity can actually be estimated as $O(n^2)$. The reason is that the divisions are performed on shorter and shorter numbers, so that we get a sum looking like $\sum_{k=0}^{n-1} n^{2(n-k)/n} \approx n^2$.

There exist almost linear ($O(n \log^2 n \log\log n)$) algorithms for computing gcds.

2.1.5.2 Memory

Also the memory requirements of the (Extended) Euclidean Algorithm are minimal. The Euclidean Algorithm requires only storage for a, b and r, i.e. 3 numbers only. The Extended Euclidean Algorithm requires additional storage for q, x_1, x_2, y_1 and y_2, as by an efficient ordering of the computations, the temporary values x_3 and y_3 can be stored in the memory space of r. This is in total 8 numbers. Even this is not yet optimal, see Exercise 2.4.

2.2 Efficient Modular Arithmetic

We now proceed with describing the basics of efficient multi-precision modular arithmetic, i.e. arithmetic in $\mathbb{Z}_m$ for some integer $m \geq 2$, not necessarily prime. In particular we deal with multi-precision numbers, including the modulus m.

2.2.1 Reduction

Reduction modulo m, also called *modular reduction*, is the process of, given an integer x and a modulus m, finding the unique number congruent to x modulo m in the complete residue system $\{0, 1, \ldots, m-1\}$. This unique number is often written as $x \pmod m$. When doing more complicated computations modulo m, modular reduction will be performed time and again. The main reason to do it is to keep the numbers small: $x \pmod m$ is never larger than the modulus, and is also unsigned. In a complicated computation reduction is often done interleaved with the computation, i.e. immediately after, or even interleaved with, each addition / subtraction / multiplication.

Reduction can often be done by common sense methods. For example, when you are certain that the number to be reduced is between m and $2m-1$, then simply subtracting m suffices. And when you are certain that it is between $-m$ and -1, simply adding m suffices.

The general method is division with remainder. So a naive algorithm is the following:

Algorithm 2.4 (Modular Reduction, naive method)

Input:	$x, m \in \mathbb{Z}$ `with` $m \geq 2$
Output:	$y \in \{0, 1, \ldots, m-1\}$ `such that` $y \equiv x \pmod m$
Step 1:	$q \leftarrow \left\lfloor \frac{x}{m} \right\rfloor$`,` $y = x - qm$`, output` y

But note that this time we're not at all interested in the quotient, only in the remainder. So an algorithm can easily be designed that does only subtractions (and shifts, i.e. multiplications by powers of the radix).

Algorithm 2.5 (Modular Reduction with radix b)

Input:	$x, m, b \in \mathbb{Z}$ `with` $m \geq 2$`,` $b \geq 2$
Output:	$y \in \{0, 1, \ldots, m-1\}$ `such that` $y \equiv x \pmod m$
Step 1:	$x' \leftarrow \lvert x \rvert$
	$k \leftarrow$ `word length of` x`,` $n \leftarrow$ `word length of` m
Step 2:	`for` $i = k - n$ `down to` 0 `do`
	$\quad$ `while` $x' \geq mb^i$ `do` $x' \leftarrow x' - mb^i$
Step 3:	`if` $x \geq 0$ `or` $x' = 0$ `then` $y \leftarrow x'$ `else` $y = m - x'$
	`output` y

This algorithm is particularly efficient when the radix is small (such as $b = 2$). Of course intermediate versions of the algorithm, doing divisions on one word numbers only, are also possible. Then

one should replace the while-loop in step 2 by one subtraction of qmb^i for the proper value of q, found by division of one- or at most two-words numbers.

Another idea to avoid the long division is at the background of *Barrett's modular reduction method*. It makes an easy to compute, educated guess of the quotient $q = \left\lfloor \frac{x}{m} \right\rfloor$. It requires a precomputation depending only on the modulus, requires $b > 3$ (which is realistic as b usually is 2 to the power the word size) and the number to be reduced less than the square of the modulus (which is realistic as the most difficult modular reduction to be performed in practice is that after multiplying two reduced numbers). It is advantageous if many modular reductions have to be done with the same modulus, as happens often in cryptology.

Algorithm 2.6 (Barrett Modular Reduction)

Input:	$x, m, b \in \mathbb{Z}$ `with` $m \geq 2$, $b \geq 3$, $m \leq x < m^2$, $n \leftarrow$ `radix` b `length of` m, `[precomputed]` $\mu \leftarrow \left\lfloor \frac{b^{2n}}{m} \right\rfloor$
Output:	$y \in \{0, 1, \ldots, m-1\}$ `such that` $y \equiv x \pmod{m}$
Step 1:	$q_0 \leftarrow \left\lfloor \frac{x}{b^{n-1}} \right\rfloor$, $q \leftarrow \left\lfloor \frac{\mu q_0}{b^{n+1}} \right\rfloor$
Step 2:	$r_1 \leftarrow x \pmod{b^{n+1}}$, $r_2 \leftarrow qm \pmod{b^{n+1}}$
Step 3:	`if` $r_1 \geq r_2$ `then` $y \leftarrow r_1 - r_2$ `else` $y \leftarrow r_1 - r_2 + b^{n+1}$
Step 4:	`while` $y \geq m$ `do` $y \leftarrow y - m$, `output` y

The precomputation has to be done only once, of course.

Note that the divisions in step 1 all are only simple shifts.

Clearly the complexity of the above algorithms, i.e. the number of operations on one word, is $O((\log x)(\log m))$. In practice one applies the algorithm so often that the inputs x are at most of the size of m^2. Then the algorithm runs in complexity $O((\log m)^2)$. In practice the algorithm is reasonably efficient for moduli of many thousands of bits.

2.2.2 Addition, subtraction and multiplication

With modular arithmetic algorithms we will always assume that the input is in reduced form. *Modular addition* and *subtraction* is straightforward.

Algorithm 2.7 (Modular Addition)

Input:	$m \in \mathbb{Z}$ `with` $m \geq 2$, $x, y \in \{0, 1, 2, \ldots, m-1\}$
Output:	$z \in \{0, 1, \ldots, m-1\}$ `such that` $z \equiv x + y \pmod{m}$
Step 1:	$z' \leftarrow x + y$
Step 2:	`if` $z' < m$ `then` $z \leftarrow z'$ `else` $z \leftarrow z' - m$ `output` z

Algorithm 2.8 (Modular Subtraction)

Input:	$m \in \mathbb{Z}$ `with` $m \geq 2$, $x, y \in \{0, 1, 2, \ldots, m-1\}$
Output:	$z \in \{0, 1, \ldots, m-1\}$ `such that` $z \equiv x - y \pmod{m}$
Step 1:	$z' \leftarrow x - y$
Step 2:	`if` $z' \geq 0$ `then` $z \leftarrow z'$ `else` $z \leftarrow z' + m$ `output` z

Modular multiplication can be done in a similar way, using any of the multiplication algorithms presented in Section 1.3.

Algorithm 2.9 (Modular Multiplication, naive method)

Input:	$m \in \mathbb{Z}$ `with` $m \geq 2$, $x, y \in \{0, 1, 2, \ldots, m-1\}$
Output:	$z \in \{0, 1, \ldots, m-1\}$ `such that` $z \equiv xy \pmod{m}$
Step 1:	`compute` $z' \leftarrow xy$ `by any convenient multiplication method`
Step 2:	`compute` $z \leftarrow z' \pmod{m}$ `by any convenient modular reduction method`
	`output` z

But here more variants are possible, interleaving the modular reduction in various ways into the multiplication algorithm. In the following algorithm modular reduction has to be done only on numbers that are at most one word longer than the modulus. This avoids both long divisions and numbers of word length larger than 1+ the word length of the modulus.

Algorithm 2.10 (Modular Multiplication with interleaved modular reduction)

Input:	$b \geq 2$, $m \in \mathbb{N}$, $m \geq 2$, $x, y \in \{0, 1, 2, \ldots, m-1\}$
Output:	$z \in \{0, 1, \ldots, m-1\}$ `such that` $z \equiv xy \pmod{m}$
Step 1:	$n \leftarrow$ `the smallest even number that is at least the length of the radix` b `representation of` m
Step 2:	$x_{\mathtt{lo}} \leftarrow x \pmod{b^{n/2}}$, $x_{\mathtt{hi}} \leftarrow (x - x_{\mathtt{lo}})b^{-n/2}$, $y_{\mathtt{lo}} \leftarrow y \pmod{b^{n/2}}$, $y_{\mathtt{hi}} \leftarrow (y - y_{\mathtt{lo}})b^{-n/2}$
Step 3:	`compute by Karatsuba's method` $z_0 \leftarrow x_{\mathtt{lo}}y_{\mathtt{lo}} \pmod{m}$, $z_1 \leftarrow x_{\mathtt{hi}}y_{\mathtt{lo}} + x_{\mathtt{lo}}y_{\mathtt{hi}} \pmod{m}$, $z_2 \leftarrow x_{\mathtt{hi}}y_{\mathtt{hi}} \pmod{m}$
Step 4:	$z \leftarrow z_2$, `for` i `from 1 to` $n/2$ `do` $z \leftarrow bz \pmod{m}$, $z \leftarrow z + z_1 \pmod{m}$, `for` i `from 1 to` $n/2$ `do` $z \leftarrow bz \pmod{m}$, $z \leftarrow z + z_0 \pmod{m}$, `output` z

2.2.3 Modular inversion and division

Modular inversion, i.e. given $a \pmod{m}$ such that a and m are coprime, computing the unique number $b \pmod{m}$ with $ab \equiv 1 \pmod{m}$, can be done by the Extended Euclidean Algorithm. Namely, this algorithm finds $x, y \in \mathbb{Z}$ such that $xa + ym = 1$, so $b \equiv x \pmod{m}$ is the answer. Indeed, we can slightly simplify the algorithm in this case, as shown below. Note that also the Extended Binary Euclidean Algorithm (see Exercise 2.6) can be adapted to produce the modular inverse efficiently.

The bit complexity of this algorithm is that of the Euclidean Algorithm, i.e. $O((\log m)^2)$.

Algorithm 2.11 (Modular Inversion)

Input:	$m \in \mathbb{Z}$ `with` $m \geq 2$, $a \in \{0, 1, \ldots, m-1\}$
Output:	$a^{-1} \pmod{m} \in \{0, 1, \ldots, m-1\}$ `if it exists, an error message otherwise`
Step 1:	$a' \leftarrow a$, $m' \leftarrow m$, $x_1 \leftarrow 1$, $x_2 \leftarrow 0$
Step 2:	`while` $m' > 0$ `do` $q \leftarrow \left\lfloor \frac{a'}{m'} \right\rfloor$, $r \leftarrow a' - qm'$, $a' \leftarrow m'$, $m' \leftarrow r$, $x_3 \leftarrow x_1 - qx_2$, $x_1 \leftarrow x_2$, $x_2 \leftarrow x_3$
Step 3:	`if` $a' = 1$ `then` $a^{-1} \leftarrow x_1$, `output` a^{-1} `else output "inverse does not exist"`

Modular division, i.e. given $a, b \pmod{m}$ such that a and m are coprime, computing the unique

number $c \pmod m$ with $ac \equiv b \pmod m$, can be done by modular inversion followed by a multiplication. However, notice that modular division sometimes is possible also when a and m are not coprime, see Exercise 2.13.

2.2.4 Exponentiation

In Section 1.3 we did not treat exponentiation for integers, as this is usually not very practical. Indeed, computing a^b for large a, b will in general be infeasible for b that is larger than 1000 or so. For example, when the numbers are only 10 decimal digits long, a^b can have more than 10^{10} decimal digits, and that's way too much to be practical.

However, when we are doing modular arithmetic, the situation is much better. Indeed, exponentiation methods exist that keep all intermediate results reasonably small, namely of the size of the modulus. And these methods are pretty efficient too, so that modular exponentiations of numbers of 1000 digits can be done in practice. Cryptographers make use of this very much.

Assume we have a modulus m, a number $x \pmod m$ and an exponent a (may all be integers with thousands of bits), and we want to compute $x^a \pmod m$.

The naive method, of multiplying x by itself $a-1$ times, and reducing after each multiplication, does keep the intermediate results small, but takes a prohibitively long time. (complexity $O(a)$, which is exponential in the size of a, where you actually would like a method that is only polynomial in the size of a.

2.2.4.1 Square and multiply

More efficient ways of performing modular exponentiation use *repeated square and multiply*. Note that repeated squaring of x gives $x^2, x^4, x^8, x^{16}, x^{32}, \ldots$, and exponents in between two powers of 2 can be found as sums of powers of 2, e.g. $22 = 16+4+2$. This means that $x^{22} = x^{16+4+2} = x^{16}x^4x^2$ can be computed with only 4 squarings and 2 multiplications. Note that the binary expansion of the exponent plays an important role here.

We now give several variants. Always we assume that the base of the exponentiation is reduced modulo m. In theory we can also assume that the exponent is reduced modulo $\phi(m)$ (the Euler Totient function), but that is not necessary for describing our algorithms, and moreover this reduction is not always possible in practice, because often the exact value of $\phi(m)$ is not known.

We start with the binary variants, where we use the binary expansion of the exponent. We can scan the binary expansion of the exponent from right to left or from left to right.

Algorithm 2.12 (Modular Exponentiation, binary right to left method)

Input:	$m, a \in \mathbb{N}$ `with` $m \geq 2$, $x \in \{0, 1, \ldots, m-1\}$
Output:	$z \in \{0, 1, \ldots, m-1\}$ `such that` $z \equiv x^a \pmod m$
Step 1:	$z \leftarrow 1$, $s \leftarrow x$, $a' \leftarrow a$
Step 2:	`while` $a' > 0$ `do`
	`if` a' `is odd then` $z \leftarrow sz \pmod m$
	$a' \leftarrow \left\lfloor \frac{a'}{2} \right\rfloor$, `if` $a' > 0$ `then` $s \leftarrow s^2 \pmod m$
Step 3:	`output` z

Scanning from left to right is illustrated by noting that $22 = ((2 \times 2 + 1) \times 2 + 1) \times 2$.

Algorithm 2.13 (Modular Exponentiation, binary left to right method)

Input:	$m, a \in \mathbb{N}$ `with` $m \geq 2$, $x \in \{0, 1, \ldots, m-1\}$, `with the binary expansion` $[a]_2 = [a_{n-1} \ldots a_0]_2$ `of` a
Output:	$z \in \{0, 1, \ldots, m-1\}$ `such that` $z \equiv x^a \pmod{m}$
Step 1:	$z \leftarrow 1$
Step 2:	`for` i `from` $n-1$ `down to` 0 `do` $z \leftarrow z^2 \pmod{m}$ `if` $a_i = 1$ `then` $z \leftarrow xz \pmod{m}$
Step 3:	`output` z

The complexity of these algorithms is not hard to estimate. The number of multiplications and squarings is at most $\left\lfloor \frac{\log a}{\log 2} \right\rfloor$, and each multiplication / squaring uses numbers less than the modulus. When the exponent a is reduced, and a modular multiplication algorithm with complexity $O((\log m)^2)$ is used (as will typically be the case), then the running time of modular exponentiation is $O((\log m)^3)$.

2.2.4.2 Montgomery

Modular multiplication as treated above is essentially not faster than first doing a multiplication, and then doing a full modular reduction by division. Montgomery reduction is a technique that avoids the costly division step, at the cost of pre- and post-computations. This is especially effective when a complicated computation has to be done involving a number of modular multiplications on intermediate numbers, all with the same modulus. The first application that comes to mind is in modular exponentiation.

Let $m \in \mathbb{N}$, $m \geq 2$ be a modulus. We now choose a fixed number R with $R > m$ and $\gcd(R, m) = 1$, and we assume that we have to compute only with numbers $x \in \mathbb{Z}$ with $0 \leq x < Rm$. In practice we take $R = b^n$ where b is the radix (word size, so usually a suitable power of 2), and n is taken such that $b^{n-1} \leq m < b^n$, i.e. essentially the word size of m Note that with an odd modulus and $R = b^n$ a power of 2, the condition $\gcd(R, m) = 1$ is automatic.

We now introduce the *Montgomery representation* $M_R(x)$ of x, and its inverse operation *Montgomery reduction* $M_R^{-1}(x)$ of x, both $\pmod{m}$ and with respect to R, as follows:

$$\begin{array}{lrcll} \text{Montgomery representation:} & M_R(x) & \equiv & xR & \pmod{m}, \\ \text{Montgomery reduction:} & M_R^{-1}(x) & \equiv & xR^{-1} & \pmod{m}. \end{array}$$

The main idea of Montgomery is to work with numbers in Montgomery representation. As precomputation $M_R(x)$ has to be computed. Furthermore a method has to be found to do efficient multiplication of numbers in Montgomery representation. Finally as postcomputation the Montgomery representation has to be undone by Montgomery reduction.

Assume that we have a procedure that for given $x, y \pmod{m}$ efficiently performs *Montgomery multiplication*, i.e. computes the Montgomery product of x and y, that is the Montgomery reduction of xy, that is

$$\text{mul}_R(x, y) \equiv M_R^{-1}(xy) \equiv xyR^{-1} \pmod{m}.$$

Applied to the Montgomery representations of x, y, we thus get

$$\text{mul}_R\left(M_R(x), M_R(y)\right) \equiv \text{mul}_R(xR, yR) \equiv (xR)(yR)R^{-1} \equiv (xyR) \equiv M_R(xy) \pmod{m}.$$

So Montgomery multiplication indeed multiplies numbers in Montgomery representation.

Note that both the Montgomery representation and the Montgomery reduction of x can be easily computed using Montgomery multiplication, because

$$M_R(x) \equiv xR \equiv \mathrm{mul}_R(x, R^2 \pmod m) \pmod m,$$
$$M_R^{-1}(x) \equiv xR^{-1} \equiv \mathrm{mul}_R(x, 1) \pmod m.$$

We will argue by example why working with the Montgomery representation may be advantageous. To compute $x^5 \mod m$, the original left-to-right exponentiation method goes as follows:

squaring:	$x_1 \leftarrow x^2$	
reduction:	$x_2 \leftarrow x_1 \pmod m$	(so $x_2 \equiv x^2 \pmod m$)
squaring:	$x_3 \leftarrow x_2^2$	
reduction:	$x_4 \leftarrow x_3 \pmod m$	(so $x_4 \equiv x^4 \pmod m$)
multiplication:	$x_5 \leftarrow x x_4$	
reduction:	$x_6 \leftarrow x_5 \pmod m$	(so $x_6 \equiv x^5 \pmod m$)

Note that three modular reductions are necessary, each requiring a costly division.

With Montgomery multiplication the computation of $x^5 \mod m$ can be done as follows:

pre-computation:	$x_1 \leftarrow \mathrm{mul}_R(x, R^2 \pmod m)$	(so $x_1 \equiv xR \pmod m$)
Montgomery square:	$x_2 \leftarrow \mathrm{mul}_R(x_1, x_1)$	(so $x_2 \equiv x_1^2 R^{-1} \equiv x^2 R \pmod m$)
Montgomery square:	$x_3 \leftarrow \mathrm{mul}_R(x_2, x_2)$	(so $x_3 \equiv x_2^2 R^{-1} \equiv x^4 R \pmod m$)
Montgomery multiply:	$x_4 \leftarrow \mathrm{mul}_R(x_1, x_3)$	(so $x_4 \equiv x_1 x_3 R^{-1} \equiv x^5 R \pmod m$)
Montgomery reduce:	$x_5 \leftarrow \mathrm{mul}_R(x_4, 1)$	(so $x_5 \equiv x^5 \pmod m$)

You should notice that only one modular reduction is necessary (in the precomputation), and that indeed all numbers are less than mR.

Montgomery multiplication (and thus reduction as well) is easy because of the following lemma.

Lemma 2.9 *Let $m, b, n \in \mathbb{N}$ satisfy $b \geq 2$, $2 \leq m < b^n$, $\gcd(m, b) = 1$. Put $R = b^n$, $m' \equiv -m^{-1} \pmod R$. Let $x, y \in \mathbb{Z}$ satisfy $0 \leq x < m$, $0 \leq y < m$.*
If $u = xym' \pmod R$ then $(xy + um)/R$ is an integer, and $(xy + um)/R \equiv \mathrm{mul}_R(x, y) \pmod m$.

Proof. Note that $xy + um \equiv xy(1 + mm') \equiv 0 \pmod R$, so $(xy + um)/R$ is an integer. Clearly $(xy + um)/R \equiv xyR^{-1} = \mathrm{mul}_R(x, y) \pmod m$. □

Note that $0 \leq (xy + um)/R < 2m$, so to compute $\mathrm{mul}_R(x, y)$ it suffices to compute $(xy + um)/R$, and if necessary, to subtract m.

Note that computing xy can be done by an efficient method such as Karatsuba's.

In practice the following algorithm is used, where the idea of Lemma 2.9 is worked out per radix b word, rather than at once for the whole $R = b^n$. This means also that $m' = -m^{-1}$ is required only modulo b.

Algorithm 2.14 (Montgomery Multiplication)

Input:	$m, b, n \in \mathbb{N}$ `with` $b \geq 2$, $2 \leq m < b^n$, $\gcd(m, b) = 1$,
	$m' = -m^{-1} \pmod b$,
	$x, y \in \mathbb{Z}$ `with` $0 \leq x < m$, $0 \leq y < m$ `and radix` b `representations`
	$[x]_b = [x_{n-1}, \ldots, x_0]_b$, $[y]_b = [y_{n-1}, \ldots, y_0]_b$
Output:	$\mathrm{mul}_{b^n}(x, y) = M_{b^n}^{-1}(xy)$
Step 1:	$a_0 \leftarrow 0$
Step 2:	`for` i `from` 0 `to` $n - 1$ `do`
	$u_i \leftarrow ((a_i \pmod b) + x_i y_0)\, m' \pmod b$
	$a_{i+1} \leftarrow (a_i + x_i y + u_i m)/b$
Step 3:	`if` $a_n \geq m$ `then` $a \leftarrow a_n - m$, `else` $a \leftarrow a_n$, `output` a

We explain why this works. Note that in the second line of step 2 we have, using $y \equiv y_0 \pmod b$ and $m \cdot m' \equiv -1 \pmod b$, that

$$a_i + x_i y + u_i m \equiv (a_i + x_i y) + (a_i + x_i y_0) m' \cdot m \equiv 0 \pmod b,$$

so a_{i+1} is an integer. Further note that

$$\begin{aligned} b^n a_n &\equiv b^{n-1} a_{n-1} + b^{n-1} x_{n-1} y \\ &\equiv b^{n-2} a_{n-2} + b^{n-2} x_{n-2} y + b^{n-1} x_{n-1} y \\ &\equiv \ldots \\ &\equiv a_0 + x_0 y + b x_1 y + b^2 x_2 y + \ldots + b^{n-2} x_{n-2} y + b^{n-1} x_{n-1} y \\ &= xy \pmod m, \end{aligned}$$

so that indeed $a \equiv xyb^{-n} \equiv M_{b^n}^{-1}(xy) \pmod m$. To explain step 3 it remains to note that $0 \le a_n < 2m$, because $a_{i+1} \le \frac{a_i}{b} + 2m\frac{b-1}{b}$, hence

$$\begin{aligned} a_1 &\le 2m\frac{b-1}{b}, \\ a_2 &\le 2m\frac{b-1}{b^2} + 2m\frac{b-1}{b}, \\ a_3 &\le 2m\frac{b-1}{b^3} + 2m\frac{b-1}{b^2} + 2m\frac{b-1}{b}, \\ &\ldots \\ a_n &\le 2m(b-1)\left(\frac{1}{b^n} + \frac{1}{b^{n-1}} + \ldots + \frac{1}{b^2} + \frac{1}{b}\right) < 2m(b-1)\left(\ldots + \frac{1}{b^2} + \frac{1}{b}\right) = 2m. \end{aligned}$$

Also note that in the first line of step 2 only single word operations are done, and that in the second line the division by b is just a shift by one word.

Finally we give the efficient method for *Montgomery exponentiation* using Montgomery multiplication. It is based on the binary left to right exponentiation method.

Algorithm 2.15 (Montgomery Exponentiation)

Input:	$m, b, n \in \mathbb{N}$ `with` $b \ge 2$, $2 \le m < b^n$, $\gcd(m, b) = 1$,
	$x \in \{0, 1, \ldots, m-1\}$,
	$a \in \mathbb{N}$ `with the binary expansion` $[a]_2 = [a_{t-1} \ldots a_0]_2$
	`[precomputed]` $r_1 = b^n \pmod m$, $r_2 = b^{2n} \pmod m$
Output:	$z \in \{0, 1, \ldots, m-1\}$ `such that` $z \equiv x^a \pmod m$
Step 1:	$z \leftarrow r_1$, $x' \leftarrow \mathrm{mul}_{b^n}(x, r_2) \pmod m$
Step 2:	`for` i `from` $t-1$ `down to` 0 `do`
	$\quad z \leftarrow \mathrm{mul}_{b^n}(z, z)$
	$\quad$ `if` $a_i = 1$ `then` $z \leftarrow \mathrm{mul}_{b^n}(x', z)$
Step 3:	$z \leftarrow \mathrm{mul}_{b^n}(z, 1)$, `output` z

Note that Step 1 does the inverse Montgomery reduction, step 2 is just a straight *Montgomerified* Algorithm 2.13, and step 3 is Montgomery reduction.

Montgomery exponentiation still is cubic (complexity $O(n^2 t)$ word multiplications, so $O(n^3)$ when $t \approx n$), but no divisions have to be done, apart from once in the precomputation. So in practice an important saving is accomplished.

2.2.4.3 Windowing

Finally a short note about *windowing* techniques. This is a variant of the binary left-to-right exponentiation method, where in stead of scanning the exponent bit by bit, one takes 'windows', i.e. a block of bits at once. The advantage is that less multiplications have to be done, the disadvantage that a precomputation is required, and storage of the precomputed numbers.

Algorithm 2.16 (Modular Exponentiation, window method)

Input:	$m \in \mathbb{N}$ `with` $m \geq 2$, $x \in \{0, 1, \ldots, m-1\}$, $k \in \mathbb{N}$,
	$a \in \mathbb{N}$ `with radix` 2^k `representation` $[a]_{2^k} = [a_{n-1} \ldots a_0]_{2^k}$
	`[precomputed]` $x^i \pmod m$ `for` $i = 2, 3, \ldots, 2^k - 1$
Output:	$z \in \{0, 1, \ldots, m-1\}$ `such that` $z \equiv x^a \pmod m$
Step 1:	$z \leftarrow 1$
Step 2:	`for` i `from` $n-1$ `down to` 0 `do`
	$\quad z \leftarrow z^{2^k} \pmod m$
	$\quad$ `if` $a_i \neq 0$ `then` $z \leftarrow x^{a_i} z \pmod m$
Step 3:	`output` z

Variants exist that e.g. optimize storage, and optimize multiplications e.g. by 'sliding windows', where one skips blocks of bits that contain too many zeroes.

2.3 The Chinese Remainder Theorem

2.3.1 Theoretic viewpoint

An important topic to study is systems of linear congruence equations, where we have only one variable, but different moduli.

As an example, let us try to solve in a naive way

$$\begin{cases} 3x \equiv 7 \pmod{11}, \\ 2x \equiv 9 \pmod{13}. \end{cases}$$

The first congruence can be solved on noting that $3^{-1} = 4 \pmod{11}$, hence $x \equiv 7 \times 4 \equiv 6 \pmod{11}$. Hence we may write $x = 6 + 11k$ for some $k \in \mathbb{Z}$. Substituting this into the second congruence we get $12 + 22k \equiv 9 \pmod{13}$, or, equivalently, $9k \equiv 10 \pmod{13}$. Note that $9^{-1} = 3 \pmod{13}$, hence $k \equiv 10 \times 3 \equiv 4 \pmod{13}$. Hence we may write $k = 4 + 13\ell$ for some $\ell \in \mathbb{Z}$. Substituting this back into $x = 6 + 11k$ we get $x = 6 + 11(4 + 13\ell) = 50 + 143\ell$, which comes down to $x \equiv 50 \pmod{143}$. Note that there is exactly one solution modulo the product of the original moduli.

In general, we want to have an efficient method for solving the following system:

$$\begin{cases} a_1 x & \equiv & b_1 \pmod{m_1}, \\ a_2 x & \equiv & b_2 \pmod{m_2}, \\ & \vdots & \\ a_k x & \equiv & b_k \pmod{m_k}. \end{cases}$$

To start with, note that in order for solutions to exist we must have $\gcd(a_i, m_i) \mid b_i$ for all i. If that condition is satisfied, then in each congruence we can divide by $\gcd(a_i, m_i)$. Hence we may assume without loss of generality that $\gcd(a_i, m_i) = 1$ for all i. Then by multiplying each congruence by $a_i^{-1} \pmod{m_i}$ we may also assume without loss of generality that $a_i = 1$ for all i.

Next assume that there exist $i \neq j$ such that $d = \gcd(m_i, m_j) > 1$. Then $x \equiv b_i \pmod{d}$ and $x \equiv b_j \pmod{d}$, so a necessary condition then is that $b_i \equiv b_j \pmod{d}$. At least one of m_i and m_j, say m_j, now satisfies $\gcd\left(d, \frac{m_j}{d}\right) = 1$, and the set of solutions to $x \equiv b_i \pmod{m_i}, x \equiv b_j \pmod{m_j}$ is then equal to the set of solutions to $x \equiv b_i \pmod{m_i}$, $x \equiv b_j \pmod{\frac{m_j}{d}}$ (exercise 2.13). Now we replace m_j by $\frac{m_j}{d}$, and then we see that we may assume without loss of generality that all moduli are pairwise coprime.

Now we are in a position to formulate the main result, which was already known to mathematicians in ancient China.

Theorem 2.10 (Chinese Remainder Theorem) *Let $m_1, m_2, \ldots m_k$ be pairwise coprime integers ≥ 2, and let $b_1, b_2, \ldots, b_k \in \mathbb{Z}$. Let $m = \prod_{i=1}^{k} m_i$. There exists exactly one solution $x \pmod{m}$ of the system*

$$\begin{cases} x & \equiv & b_1 \pmod{m_1}, \\ x & \equiv & b_2 \pmod{m_2}, \\ & \vdots & \\ x & \equiv & b_k \pmod{m_k}. \end{cases}$$

Proof. To show existence, we follow Gauss' method. Put $\mu_i = \frac{m}{m_i} = \prod_{\substack{j=1 \\ j \neq i}}^{k} m_i$, and let $n_i = \mu_i^{-1} \pmod{m_i}$ (note that this inverse exists because all moduli are coprime). Then $\mu_j n_j \equiv \begin{cases} 0 \pmod{m_i} & \text{if } i \neq j \\ 1 \pmod{m_i} & \text{if } i = j \end{cases}$, and so $x = \sum_{j=1}^{k} b_j \mu_j n_j$ is a solution.
To show uniqueness, let us take two solutions x, x'. Then $m_i \mid x - x'$ for all i, hence[1] $\operatorname{lcm}(m_1, m_2, \ldots, m_k) \mid x - x'$. But as $\gcd(m_1, m_2, \ldots, m_k) = 1$ we must have (by easily proved properties of the lcm) that $\operatorname{lcm}(m_1, m_2, \ldots, m_k) = m$. Hence $x \equiv x' \pmod{m}$. □

Note that Gauss' method, as described in the proof above, can in principle be used to compute the solution. The complexity is $O((\log m)^2)$ bit operations.

Example: with $x \equiv 6 \pmod{11}$, $x \equiv 11 \pmod{13}$, $x \equiv 16 \pmod{17}$ we get $\mu_1 = 221$, $\mu_2 = 187$, $\mu_3 = 143$, $n_1 = 1$, $n_2 = 8$, $n_3 = 5$, so $x = 6 \times 221 \times 1 + 11 \times 187 \times 8 + 16 \times 143 \times 5 = 29222 \equiv 50 \pmod{2431}$.

2.3.2 Algorithmic viewpoint

A more efficient method is *Garner's Algorithm*, especially when a number of systems have to be solved with the same set of moduli. The main advantage is that only reductions modulo m_i are required.

A case of particular importance for cryptography is the case of $x \equiv b_p \pmod{p}$, $x \equiv b_q \pmod{q}$, where p, q are distinct primes. The abbreviation *CRT* is used in the cryptographic literature a lot, and it usually refers to this special case. The solution is then often given as follows:

$$u \leftarrow p^{-1} \pmod{q},$$
$$x \leftarrow b_p + (b_q - b_p)up \pmod{pq}.$$

[1] $\operatorname{lcm}(a, b)$ is the *least common multiple* of a and b, which is equal to $ab/\gcd(a, b)$.

Algorithm 2.17 (Garner's Algorithm)

Input:	$m_1, m_2, \ldots, m_k \in \mathbb{Z}$ `with` $m_i \geq 2$ `pairwise coprime,` $b_1, b_2, \ldots, b_k \in \mathbb{Z}$`,` `[precomputed]` $C_i = \prod_{j=1}^{i-1} m_j^{-1} \pmod{m_i}$ `for all` $i = 2, 3, \ldots, k$
Output:	x `such that` $x \equiv m_i \pmod{m_i}$ `for all` $i = 1, 2, \ldots, k$`,` `with` $x \in \{0, 1, \ldots, m-1\}$`, where` $m = \prod_{i=1}^{k} m_i$
Step 1:	$x \leftarrow b_1$
Step 2:	`for` i `from 2 to` k `do` $u \leftarrow (b_i - x)C_i \pmod{m_i}$ $x \leftarrow x + u \prod_{j=1}^{i-1} m_j$
Step 3:	`output` x

2.3.3 Practical applications

The Chinese Remainder Theorem has many applications, of which we mention a few practical ones.

In RSA cryptography it is used to speed up the private key operation. See any introductory course on public key cryptography.

In [Kn, Section 4.3.3B] a method from Schönhage, different from the generalized Karatsuba method described in Section 1.3.2.3, is described in which modular arithmetic and the Chinese Remainder Theorem are used to speed up multiplication of large numbers to almost linear complexity.

In [Sh, Section 4.4] another application of the Chinese Remainder Theorem is presented that shows how sometimes specific computations can be sped up.

Exercises

2.1. Use the Extended Euclidean Algorithm to compute $\gcd(3507, 2181)$, and to find $x, y \in \mathbb{Z}$ such that $3507x + 2181y = \gcd(3507, 2181)$.

2.2. Let $a, b \in \mathbb{Z}$ and $d = \gcd(a, b)$. Theorem 2.2 shows the existence of $x_0, y_0 \in \mathbb{Z}$ such that $d = x_0 a + y_0 b$. Find all solutions (x, y) to $d = xa + yb$, i.e. express them in terms of (x_0, y_0).

2.3. Let $d = \gcd(a, b)$. Show that $\gcd\left(\frac{a}{d}, \frac{b}{d}\right) = 1$.

2.4. Find a way to implement the Extended Euclidean Algorithm with only enough memory for 7 numbers.

2.5. Write out a proof of Lemma 2.6, and of Corollary 2.7.

2.6. Devise an *Extended Binary Euclidean Algorithm*.

2.7. Compute again $\gcd(3507, 2181)$ (see Exercise 2.1), but this time using the Binary Euclidean Algorithm. If you did Exercise 2.6), also find $x, y \in \mathbb{Z}$ such that $3507x + 2181y = \gcd(3507, 2181)$ by the Extended Binary Euclidean Algorithm. Try to compare the workload (number of iterations) to the original Euclidean Algorithm.

2.8. Let $a = 2^n + 1$ and $b = 2^n - 1$. Compute the number of iterations performed when using the

Binary Euclidean Algorithm to compute $\gcd(a, b)$.

2.9. (a) Give a proper definition of the greatest common divisor of three integers a, b, c.
(b) Show that three numbers can be relatively prime even if any two of them are not relatively prime.
(c) Show that $\gcd(a, b, c) = \gcd(\gcd(a, b), c)$.
(d) Show that there exist $x, y, z \in \mathbb{Z}$ such that $xa + yb + zc = \gcd(a, b, c)$.
(e) Devise an efficient algorithm to compute $\gcd(a, b, c)$.
(f) Generalize to the greatest common divisor of n integers $a_1, a_2, \ldots, a_n$.

2.10. Implement all algorithms given in Section 2.1, and the one you did in Exercise 2.6, in your favourite computer language or computer algebra system.

2.11. Prove that Barrett modular reduction (Algorithm 2.6) is correct. Hint: show that $0 \leq x - qm < 3m$, and that at the end of step 3 it is always true that $z = x - qm$.

2.12. Perform Algorithms 2.4, 2.5 and 2.6 on $x = 3507449$, $m = 1913$, $b = 10$. You can do this by hand, or write a program in your favourite language or computer algebra package.

2.13. Let $a, b, m \in \mathbb{Z}$ with $m \geq 2$. Let $d = \gcd(a, m)$. Show that $ax \equiv b \pmod{m}$ has solutions if and only if $d \mid b$, and that the solutions then are given by $x \equiv \left(\frac{a}{d}\right)^{-1} \frac{b}{d} \left(\mathrm{mod}\ \frac{m}{d}\right)$. Show that there are exactly d solutions modulo m, and describe them.

2.14. Compute the product of 3507 and 2181 modulo $m = 72639$ by Algorithm 2.14 (Montgomery multiplication) using $b = 10$, $n = 5$.

2.15. Compute $283^{55} \pmod{1001}$ by the binary right to left and left to right exponentiation algorithms.

2.16. In some cryptographic computations several modular exponentiations with the same modulus have to be performed, to produce one outcome, e.g. the computation of $x^a y^b \pmod{m}$. Devise an efficient algorithm that outputs $x^a y^b \pmod{m}$ on input of m, x, y, a, b.
Hint: adapt the binary left to right modular exponentiation algorithm. Your algorithm should be able to compute $x^{22} y^{13} \pmod{m}$ in at most 4 multiplications and 4 squarings, after a precomputation of 1 multiplication.

2.17. Prove the correctness of Garner's Algorithm 2.17.

2.18. Solve $x \equiv 2 \pmod{5}$, $x \equiv 1 \pmod{7}$, $x \equiv 3 \pmod{11}$, $x \equiv 8 \pmod{13}$, both by Gauss' method and Garner's algorithm.

2.19. Find all $x \in \mathbb{Z}$ that satisfy $x \equiv 3 \pmod{8}$, $x \equiv 7 \pmod{15}$ and $x \equiv 12 \pmod{25}$.

2.20. Implement all algorithms given in Section 2.2 in your favourite computer language or computer algebra system.

2.21. Implement Gauss' and Garner's methods given in Section 2.3 in your favourite computer language or computer algebra system.

Chapter 3

Multiplicative structure of $\mathbb{Z}_n^*$

Introduction

General references for this chapter: [BS, Chapters 4, 5], [CP, Chapter 9], [GG, Chapters 3, 4, 5], [Sh, Chapters 4, 11]. And for those preferring Dutch: [Be, Chapter 7], [Ke, Chapter 11], [dW, Chapter 2].

In this Chapter we study the multiplicative properties of the integers modulo n. In particular the following concepts will be treated:

- order of an element,
- primitive root.

3.1 Euler (and Fermat)

We start with repeating the required concepts and results from elementary algebra that have been studied in the first year Algebra course.

By $\mathbb{Z}_n$ we denote the integers modulo n, for a given integer $n \geq 2$, not necessarily prime. Usually we denote the elements of $\mathbb{Z}_n$ by $0, 1, \ldots, n-1$. Formally speaking all this is abuse of notation, since the elements are not numbers, but residue classes, but confusion should not arise.

A set $\{a_1, a_2, \ldots, a_k\} \subset \mathbb{Z}$ is called a *reduced residue system* (mod n) if it contains exactly one representative for each congruence class of integers that are coprime to n. A reduced residue system modulo n has exactly $\phi(n)$ elements, where $\phi(n)$ is Euler's Totient function. This function has the following properties.

Theorem 3.1 (Euler ϕ function)
(a) *ϕ is a multiplicative function, i.e. if* $\gcd(m,n) = 1$ *then* $\phi(mn) = \phi(m)\phi(n)$.
(b) *If* $n = \prod_{i=1}^{k} p_i^{e_i}$ *is the factorization of n into primes, then*

$$\phi(n) = \prod_{i=1}^{k} p_i^{e_i - 1}(p_i - 1) = n \prod_{i=1}^{k} \left(1 - \frac{1}{p_i}\right).$$

Proof. **(a)** Consider the map $\rho : \mathbb{Z}_{mn}^* \to \mathbb{Z}_m^* \times \mathbb{Z}_n^*$ defined by $\rho(a \pmod{mn}) = (a \pmod{m}, a \pmod{n})$. It suffices to show that this map is bijective. That it is surjective is just the Chinese Remainder Theorem (Theorem 2.10). To show that it is injective, let $a, b \in \mathbb{Z}_{mn}^*$ be such that $\rho(a) = \rho(b)$. Then $a \equiv b \pmod{m}$ and $a \equiv b \pmod{n}$, so $m|a-b$ and $n|a-b$, and because m and n are relatively prime it follows that $mn|a-b$, hence $a \equiv b \pmod{mn}$.
(b) In view of (a) it suffices to prove that $\phi(p^e) = p^{e-1}(p-1)$ for any prime p and any $e \geq 1$. Note that there are p^e elements in $\{0, 1, 2, \ldots, p^e - 1\}$, and exactly p^{e-1} of them are divisible by p, namely $0, p, 2p, \ldots, p^e - p$. So $\phi(p^e) = p^e - p^{e-1} = p^{e-1}(p-1)$. □

One of the reasons why Euler's $\phi(n)$ is important is the following result.

Theorem 3.2 (Euler's Theorem) *If $n \geq 2$ and $a \in \mathbb{Z}$ coprime to n, then $a^{\phi(n)} \equiv 1 \pmod{n}$.*

Proof. Let $\mathbb{Z}_n^* = \{a_1, a_2, \ldots, a_{\phi(n)}\}$, and let $a \in \mathbb{Z}_n^*$. Note that if $aa_i \equiv aa_j \pmod{n}$, then the fact that $\gcd(a, n) = 1$ implies that $a_i \equiv a_j \pmod{n}$, and hence $aa_1, aa_2, \ldots, aa_{\phi(n)}$ are all different $\pmod{n}$. This implies $\mathbb{Z}_n^* = \{aa_1, aa_2, \ldots, aa_{\phi(n)}\}$. Multiply all elements of $\mathbb{Z}_n^*$ in two ways: $\prod_{i=1}^{\phi(n)} a_i \equiv \prod_{i=1}^{\phi(n)} aa_i = a^{\phi(n)} \prod_{i=1}^{\phi(n)} a_i \pmod{n}$, and dividing out $\prod_{i=1}^{\phi(n)} a_i$ the result follows. □

An obvious special case of Euler's theorem is Fermat's theorem.

Theorem 3.3 (Fermat's Theorem) *If p is prime and $a \in \mathbb{Z}$ such that $p \nmid a$, then $a^{p-1} \equiv 1 \pmod{p}$.*

Note that for some a there may exist smaller exponents than $p-1$ for which the power is already 1 modulo p.

An important consequence of Euler's theorem is that when computing powers modulo n, we can take exponents modulo $\phi(n)$. Another consequence is that we can compute modular inverses by modular exponentiation.

Corollary 3.4 *Let $n \geq 2$ and $x \in \mathbb{Z}$ coprime to n.*
(a) *$x^a \equiv x^b \pmod{n}$ if $a \equiv b \pmod{\phi(n)}$.*
(b) *The inverse x^{-1} of x modulo n is given by $x^{-1} \equiv x^{\phi(n)-1} \pmod{n}$.*

The next result is easy to prove, see Exercise 3.3.

Theorem 3.5 $\sum_{d|n} \phi(d) = n$.

We will also need a few results on the number of solutions of polynomial congruences.

Theorem 3.6 *Let p be prime, and let $f \in \mathbb{Z}_p[X]$ be a polynomial of degree n. Then $f(x) \equiv 0 \pmod{p}$ has at most n different solutions in $\mathbb{Z}_p$.*

Proof. For $n = 1$ this is trivial. For $n > 1$ we apply induction. Suppose any polynomial in $\mathbb{Z}_p[X]$ of degree $\leq n-1$ has at most $n-1$ zeroes. When f has no zeroes, then the result is trivially true, so we assume that f has at least one zero, say α. By division with remainder, we see that there exists $q \in \mathbb{Z}_p[X]$ such that $f(X) = (X - \alpha)q(X)$, and clearly q has degree $n-1$. Any zero of f different from α now is a zero of q (note that here we use the fact that p is prime; for composite p this is not true anymore). Induction now shows that the total number of zeroes of f is $\leq 1 + (n-1)$. □

Theorem 3.7 *Let p be prime, and $d \mid p-1$. Then $x^d - 1 \equiv 0 \pmod{p}$ has exactly d solutions.*

Proof. Fermat's Theorem 3.3 shows that $x^{p-1} - 1 \equiv 0 \pmod{p}$ has exactly $p-1$ solutions. Now write $p - 1 = kd$. Note that

$$x^{p-1} - 1 = (x^d - 1)\left(x^{(k-1)d} + x^{(k-2)d} + \ldots + x^d + 1\right),$$

and Theorem 3.6 shows that the total number of zeroes is $(\leq d) + (\leq (k-1)d)$. The result now follows at once. □

3.2 Order of an element

We know that $\mathbb{Z}_n$ is a commutative ring. It follows that $\mathbb{Z}_n^* = \{x \in \mathbb{Z} | 0 \leq x \leq n-1, \gcd(x,n) = 1\}$ is a multiplicative group. Note that $\#\mathbb{Z}_n^* = \phi(n)$. In particular, when p is prime then $\mathbb{Z}_p^* = \{1, 2, \ldots, p-1\}$ (in this case $\mathbb{Z}_p$ is even a field).

In group theory, the term "order" is used in two different ways.

In the first place the number of elements of a finite group G (such as $\mathbb{Z}_n^*$) is called the *order of the group*, and we use the notation $\mathrm{ord}(G) = \#G$.

In the second place each element of a *finite* group has an order. This is defined as follows: if G is a finite group and $a \in G$, then the *order of the element* a is the smallest positive integer e such that $a^e = 1$. Notation: $\mathrm{ord}(a)$.

In the case of $G = \mathbb{Z}_n^*$ the equation $a^e = 1$ really means $a^e \equiv 1 \pmod{n}$. For example, in Z_7^* we have $\mathrm{ord}(1) = 1$, $\mathrm{ord}(6) = 2$, $\mathrm{ord}(2) = \mathrm{ord}(4) = 3$, and $\mathrm{ord}(3) = \mathrm{ord}(5) = 6$.

Note that the cyclic subgroup of G generated by a, which is denoted by $\langle a \rangle = \{1, a, a^2, a^3, \ldots\}$, has exactly $\mathrm{ord}(a)$ elements, i.e. $\mathrm{ord}(\langle a \rangle) = \mathrm{ord}(a)$, so the two concepts of order coincide in this case.

We now have the following results, giving useful properties of element orders, notably that the order of an element always divides the order of the group it lies in.

Lemma 3.8 *Let $a \in \mathbb{Z}_n^*$.*
(a) $\mathrm{ord}(a)$ *exists.*
(b) *If $e \geq 1$ such that $a^e \equiv 1 \pmod{n}$ then $\mathrm{ord}(a) \mid e$.*
(c) $\mathrm{ord}(a) \mid \phi(n)$.

Proof. (a) Euler's Theorem 3.2 shows that there exists some $e \geq 1$ with $a^e \equiv 1 \pmod{n}$, namely $\phi(n)$. But then there also exists a smallest such e.
(b) Apply division with remainder: there exist $q, r \in \mathbb{Z}$ with $e = q\mathrm{ord}(a) + r$ and $0 \leq r < \mathrm{ord}(a)$. Clearly $a^r \equiv a^{e - q\mathrm{ord}(a)} \equiv a^e (a^{\mathrm{ord}(a)})^{-q} = 1 \cdot 1^{-q} = 1$. By the definition of $\mathrm{ord}(a)$ this means that $r = 0$.
(c) This uses (b) together with Euler's Theorem 3.2. □

The set $\langle a \rangle$ generated by a is a cyclic subgroup of $\mathbb{Z}_n^*$, with $\mathrm{ord}(\langle a \rangle) = \mathrm{ord}(a) \mid \phi(n)$. Often this order is strictly smaller than $\phi(n)$. The integer $\dfrac{\phi(n)}{\mathrm{ord}(a)}$ is sometimes called the *cofactor* of a.

Next we count the number of elements in $\mathbb{Z}_p^*$ of a given order d, but for primes p only.

Lemma 3.9 *Let p be prime, and let $d \mid p-1$. There are exactly $\phi(d)$ elements in $\mathbb{Z}_p^*$ of order d.*

Proof. Let us write

$$A_d = \#\{b \in \mathbb{Z}_p^* \mid \mathrm{ord}(b) = d\}.$$

We use induction to prove $A_d = \phi(d)$. For $d = 1$ the result is trivial, since 1 is the only element of order 1. Now assume the result holds for all $d' < d$. Theorem 3.7 and Lemma 3.8(b) tell us:

$$d = \#\{b \in \mathbb{Z}_p^* \mid b^d \equiv 1 \pmod p\} = \sum_{d'|d} A_{d'} = A_d + \sum_{d'|d, d' \neq d} A_{d'}.$$

By induction we find $d = A_d + \sum_{d'|d, d' \neq d} \phi(d')$, and then Theorem 3.5 yields $d = A_d + (d - \phi(d))$. This proves the result. See also Section 6.3. □

3.3 Primitive roots

In this section we study the multiplicative structure of the set of integers modulo n.

Sometimes the group $\mathbb{Z}_n^*$ itself is cyclic. Take for example $n = 7$. Then we have 3 as a generator, as $\{3^0, 3^1, 3^2, 3^3, 3^4, 3^5\} = \{1, 3, 2, 6, 4, 5\} = \mathbb{Z}_7^*$.

A generator of $\mathbb{Z}_n^*$ is also called a *primitive root* modulo n. In other words, a primitive root is by definition an element of order $\phi(n)$.

Sometimes a primitive root does not exist. Take for example $n = 15$. Then a simple computation shows that for all $x \in \mathbb{Z}_{15}^*$ already $x^4 \equiv 1 \pmod{15}$, whereas $\#\mathbb{Z}_{15}^* = 8$.

First we study the prime case.

Theorem 3.10 *If p is prime then $\mathbb{Z}_p^*$ is cyclic, i.e. has a primitive root.*

Proof. Lemma 3.9 tells us that there are $\phi(p-1)$ elements of order $p - 1 = \phi(p)$. □

Well, that was easy. Note however that this proof is not constructive. Next we study the case of odd prime powers.

Theorem 3.11 *If p is an odd prime and $k \geq 1$ then $\mathbb{Z}_{p^k}^*$ is cyclic, i.e. has a primitive root.*

This is more difficult. We first have two lemmas.

Lemma 3.12 *Let p be prime and $a \equiv 1 \pmod{p^t}$ for some $t \geq 1$.*
(a) *Then $a^p \equiv 1 \pmod{p^{t+1}}$.*
(b) *Suppose $p > 2$ or $t > 1$. If $a \not\equiv 1 \pmod{p^{t+1}}$ then $a^p \not\equiv 1 \pmod{p^{t+2}}$.*

Proof. Write $a = 1 + rp^t$. Note that $a^p = 1 + \binom{p}{1} rp^t + \binom{p}{2} r^2 p^{2t} + \ldots$. Because $\binom{p}{1} = p$ and $2t \geq t + 1$, (a) follows immediately. When $p > 2$ then $p \mid \binom{p}{2}$, and when $t > 1$ then $2t \geq t + 2$. In both cases we have $a^p \equiv 1 + rp^{t+1} \pmod{p^{t+2}}$. As $p \nmid r$, (b) follows. □

Lemma 3.13 *Let $a_1, a_2 \in \mathbb{Z}_n^*$ with $m_1 = \mathrm{ord}(a_1), m_2 = \mathrm{ord}(a_2)$. Assume that $\gcd(m_1, m_2) = 1$. Then $\mathrm{ord}(a_1 a_2) = m_1 m_2$.*

Proof. Let $t = \text{ord}(a_1a_2)$. On the one hand $(a_1a_2)^{m_1m_2} = (a_1^{m_1})^{m_2}(a_2^{m_2})^{m_1} \equiv 1 \pmod{n}$, so $t \mid m_1m_2$. On the other hand, note that $1 \equiv (a_1a_2)^{tm_2} = a_1^{tm_2}(a_2^{m_2})^t \equiv a_1^{tm_2} \pmod{n}$, so that $m_1 = \text{ord}(a_1) \mid tm_2$, hence $m_1 \mid t$, and similarly we find $m_2 \mid t$. Thus also $m_1m_2 \mid t$. □

Proof of Theorem 3.11. Theorem 3.10 covers the case $k = 1$, so we assume $k \geq 2$.
We are looking for an element of order $\phi(p^k) = p^{k-1}(p-1)$. We will construct such an element, by first finding one of order p^{k-1}, then constructing one of order $p-1$, and then we apply Lemma 3.13.
We first show that $\text{ord}(p+1) = p^{k-1}$. Namely, applying Lemma 3.12(a) for $t = 1, 2, 3, \ldots, k$ we get $p+1 \equiv 1 \pmod{p^1} \Rightarrow (p+1)^p \equiv 1 \pmod{p^2} \Rightarrow (p+1)^{p^2} \equiv 1 \pmod{p^3} \Rightarrow \ldots \Rightarrow (p+1)^{p^{k-1}} \equiv 1 \pmod{p^k}$, so $\text{ord}(p+1) \mid p^{k-1}$ by Lemma 3.8(b). Then, applying Lemma 3.12(b) for $t = 1, 2, 3, \ldots, k-1$ we get $p+1 \not\equiv 1 \pmod{p^2} \Rightarrow (p+1)^p \not\equiv 1 \pmod{p^3} \Rightarrow \ldots \Rightarrow (p+1)^{p^{k-2}} \not\equiv 1 \pmod{p^k}$, so $\text{ord}(p+1) = p^{k-1}$.
Now let g_1 be a primitive root $\pmod{p}$, which exists due to Theorem 3.10, and let $\ell = \text{ord}(g_1)$. Lemma 3.8(c) says $\ell \mid \phi(p^k) = p^{k-1}(p-1)$. On the other hand, $g_1^\ell \equiv 1 \pmod{p^k}$, so $g_1^\ell \equiv 1 \pmod{p}$, and because g_1 is a primitive root modulo p, Lemma 3.8(b) says $p-1 \mid \ell$. It follows that $\ell = (p-1)p^s$ for some s satisfying $0 \leq s \leq k-1$.
We now take $g_2 = g_1^{p^s}$, and we show that $\text{ord}(g_2) = p-1$. Indeed, $h = p-1$ is the smallest positive integer such that $g_2^h = g_1^{hp^s} \equiv 1 \pmod{p^k}$.
Finally we take $g = (p+1)g_2$. Lemma 3.13 now implies $\text{ord}(g) = p^{k-1}(p-1) = \phi(p^k)$, in other words: g is a primitive root. □

Note that this proof is constructive, once a primitive root $\pmod{p}$ is known.

Next we look at powers of 2.

Theorem 3.14 *$\mathbb{Z}_{2^k}^*$ is cyclic, i.e. has a primitive root, only for $k = 1, 2$.*

Proof. For $k = 1, 2$ this is trivial: 1 is a primitive root $\pmod{2}$, and 3 is a primitive root $\pmod{4}$. So assume $k \geq 3$. Any odd integer a satisfies $a^2 \equiv 1 \pmod{2^3}$. Applying Lemma 3.12(a) for successively $t = 3, 4, \ldots$ we get $a^2 \equiv 1 \pmod{2^3} \Rightarrow a^{2^2} \equiv 1 \pmod{2^4} \Rightarrow a^{2^3} \equiv 1 \pmod{2^5} \Rightarrow \ldots \Rightarrow a^{2^{k-2}} \equiv 1 \pmod{2^k}$, showing that for any $a \in \mathbb{Z}_{2^k}^*$ we have $\text{ord}(a) \leq 2^{k-2}$. As $\phi(2^k) = 2^{k-1}$ this implies that primitive roots do not exist. □

Finally we get our main result.

Theorem 3.15 (Primitive Roots) *$\mathbb{Z}_n^*$ is cyclic, i.e. has a primitive root, exactly when $n = 1, 2, 4, p^k$ or $2p^k$, where p is any odd prime, and $k \geq 1$.*
The number of primitive roots, if they exist, is $\phi(\phi(n))$.

Proof. Theorems 3.11 and 3.14 cover the prime power cases.
The case of $2p^k$ is easy. Let g_1 be a primitive root $\pmod{p^k}$. If g_1 is odd we take $g = g_1$, otherwise we take $g = g_1 + p^k$. Then g is odd, so can be thought of as an element of $\mathbb{Z}_{2p^k}^*$. Now, if $g^h \equiv 1 \pmod{2p^k}$ then $g^h \equiv 1 \pmod{p^k}$, hence $g_1^h \equiv 1 \pmod{p^k}$, hence $\phi(p^k) \mid h$. Note that $\phi(2p^k) = \phi(p^k)$, so we find $\phi(2p^k) \mid h$, implying that g is a primitive root.
Now all remaining cases can be covered as follows. When $n \neq 2^k, p^k$ or $2p^k$, then n can be factored as $n = n_1n_2$, with n_1 and n_2 both > 2, and coprime. We now use the fact that $\phi(m)$ is even for all $m > 2$. Let g be any element of $\mathbb{Z}_n^*$. Then $g^{\phi(n_1)} \equiv 1 \pmod{n_1}$ and $g^{\phi(n_2)} \equiv 1 \pmod{n_2}$, hence $g^{\text{lcm}(\phi(n_1),\phi(n_2))} \equiv 1 \pmod{n}$. We find $\text{ord}(g) \leq \text{lcm}(\phi(n_1), \phi(n_2)) = \dfrac{\phi(n_1)\phi(n_2)}{\gcd(\phi(n_1), \phi(n_2))} \leq \frac{1}{2}\phi(n)$, and clearly g cannot be a primitive root.
The number of primitive roots is found in Exercise 3.6. □

3.4 Algorithms

In the previous sections we treated the theory of primitive roots and orders. Now we will discuss practical ways of computing them.

The following lemma is useful.

Lemma 3.16 *Let $a \in \mathbb{Z}_n^*$. Then a is a primitive root if and only if $a^{\phi(n)/p} \not\equiv 1 \pmod{n}$ for all primes $p \mid \phi(n)$.*

Proof. If a is a primitive root then $a^e \not\equiv 1 \pmod{n}$ for all $1 \leq e < \phi(n)$, so certainly not for $e = \phi(n)/p$. If a is not a primitive root then its cofactor is > 1, hence has a prime divisor p. It follows that $\phi(n)/p$ is a multiple of $\mathrm{ord}(a)$, hence $a^{\phi(n)/p} \equiv 1 \pmod{n}$. □

Computing the order of an element in $\mathbb{Z}_n^*$ can very naively be done by simply computing all powers until 1 is met (or $n - 1 \equiv -1$, then you know you're exactly halfway, see Exercise 3.8). This is completely out of the question for moduli n that become bigger than a few thousand.

For large moduli n computing orders is only practically possible when $\phi(n)$ and its prime factorization are completely known. Here is a method.

Algorithm 3.1 (Order of an element of $\mathbb{Z}_n^*$)

Input:	$a, n \in \mathbb{Z}$ `with` $n \geq 2$ `and` $\gcd(a, n) = 1$,
	$P = \{p \mid p$ `prime divisor of` $\phi(n)\}$
Output:	`the order` $\mathrm{ord}(a)$ `of` $a \in \mathbb{Z}_n^*$
Step 1:	$m \leftarrow \phi(n)$
Step 2:	`for all` $p \in P$ `do`
	`while` $p \mid m$ `and` $a^{m/p} \equiv 1 \pmod{n}$ `do`
	$m \leftarrow m/p$
Step 3:	`output` m

To find elements of a given order, including primitive roots, no really clever ideas are known. Basically one proceeds with trial and error. The only trick that we use in the algorithm below is that when we happen to have found an element of order being a multiple of the requested order, we can simply take the proper power of that element.

Algorithm 3.2 (Finding an element of $\mathbb{Z}_n^*$ with given order)

Input:	$n, m \in \mathbb{Z}$ `with` $n \geq 2$ `and` $m \geq 1$, $m \mid \phi(n)$,
	$P = \{p \mid p \mid \phi(n), p$ `prime`$\}$
Output:	$a \in \mathbb{Z}_n^*$ `with` $\mathrm{ord}(a) = m$
Step 1:	$b \leftarrow 1$
Step 2:	`while` $m \nmid \mathrm{ord}(b)$ `do`
	`pick a new` b `(at random, or enumerating` $2, 3, \ldots$`)`
	`compute` $\mathrm{ord}(b)$
Step 3:	$a \leftarrow b^{\mathrm{ord}(b)/m}$, `output` a

This is a somewhat problematic algorithm, as it may decide that an element of the requested order does not exist only by exhausting the complete set $\mathbb{Z}_n^*$.

For finding primitive roots, apply the above algorithm with $m = \phi(n)$, and of course only if n is a power of an odd prime, or twice a power of an odd prime. In practice this algorithm then turns out to be pretty efficient.

Exercises

3.1. Make addition and multiplication tables for $\mathbb{Z}_7$ and $\mathbb{Z}_{15}^*$.

3.2. Compute $\phi(n)$ for $n = 1, 2, \ldots, 25$.

3.3. Let $d \mid n$. How many $a \in \{1, 2, \ldots, n\}$ have $\gcd(a, n) = d$? (Hint: use Exercise 2.3.) Conclude that $\sum_{d|n} \phi(d) = n$.

3.4. Let $n \geq 2$, and let a be coprime to n. Prove the following: $a^k \equiv a^\ell \pmod{n}$ if and only if $\mathrm{ord}(a) | k - \ell$.

3.5.
(a) Find all primitive roots modulo 9 and modulo 11.
(b) Show that 14 is a primitive root $\pmod{29}$.
(c) Find (using (b) and the proof of Theorem 3.11) a primitive root $\pmod{841}$ $(841 = 29^2)$.

3.6. Let g be a primitive root modulo n. Show that g^a is a primitive root modulo n if and only if $\gcd(a, \phi(n)) = 1$. Then show that the number of primitive roots $\pmod{n}$ is $\phi(\phi(n))$.

3.7. If $\mathrm{ord}(a) = n$ and $d \mid n$ then show that $\mathrm{ord}(a^d) = n/d$.

3.8. Let $a \in \mathbb{Z}_n^*$ with $n > 2$. Assume that there exists a positive integer e such that $a^e \equiv -1 \pmod{n}$. Then show that $\frac{1}{2}\mathrm{ord}(a)$ is the smallest such number.

3.9. A set $\{A_1, A_2, \ldots, A_k\}$ in $\mathbb{Z}_n^*$ is called a *basis* if every element of $\mathbb{Z}_n^*$ can be written in exactly one way as $A_1^{t_1} A_2^{t_2} \cdots A_k^{t_k}$, with $0 \leq t_i < \mathrm{ord}(A_i)$.
(a) Show that if a primitive root exists, it constitutes a basis on its own.
(b) Let n be not divisible by 8, and let $n = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$ be its factorization into primes. Let g_i be a primitive root $\pmod{p_i^{e_i}}$ for $i = 1, 2, \ldots, k$. Prove that a basis is given by

$$A_i \equiv \begin{cases} g_i & \pmod{p_i^{e_i}} \\ 1 & \pmod{p_j^{e_j}} \quad \text{for all} \quad j \neq i \end{cases} \qquad (i = 1, 2, \ldots, k).$$

(c) Let n be divisible by 8, and let $n = 2^{e_1} p_2^{e_2} \cdots p_k^{e_k}$ be its factorization into primes, so with $e_1 \geq 3$. Let g_i be a primitive root $\pmod{p_i^{e_i}}$ for $i = 2, 3, \ldots, k$. Prove that a basis is given by

$$A_0 \equiv \begin{cases} -1 & \pmod{2^{e_1}} \\ 1 & \pmod{p_j^{e_j}} \quad \text{for all} \quad j \geq 2 \end{cases},$$

$$A_1 \equiv \begin{cases} 5 & \pmod{2^{e_1}} \\ 1 & \pmod{p_j^{e_j}} \quad \text{for all} \quad j \geq 2 \end{cases},$$

$$A_i \equiv \begin{cases} 1 & \pmod{2^{e_1}} \\ g_i & \pmod{p_i^{e_i}} \\ 1 & \pmod{p_j^{e_j}} \quad \text{for all} \quad j \geq 2, j \neq i \end{cases} \qquad (i = 2, 3, \ldots, k).$$

(d) Let n have $n = 2^{e_1} p_2^{e_2} \cdots p_k^{e_k}$ as its factorization into primes of n, with $e_1 \geq 0$, and $e_i > 0$ for $i > 1$. Show that

$$\mathbb{Z}_n^* = \begin{cases} \mathbb{Z}_{p_2^{e_2}}^* \times \cdots \times \mathbb{Z}_{p_k^{e_k}}^* & \text{if} \quad e_1 = 0, 1 \\ \mathbb{Z}_2^* \times \mathbb{Z}_{p_2^{e_2}}^* \times \cdots \times \mathbb{Z}_{p_k^{e_k}}^* & \text{if} \quad e_1 = 2 \\ \mathbb{Z}_2^* \times \mathbb{Z}_{2^{e_1-2}}^* \times \mathbb{Z}_{p_2^{e_2}}^* \times \cdots \times \mathbb{Z}_{p_k^{e_k}}^* & \text{if} \quad e_1 \geq 3 \end{cases}.$$

Chapter 4

Quadratic Reciprocity

Introduction

General references for this chapter: [BS, Sections 5.7, 5.8, 5.9], [CP, Section 2.3], [Sh, Chapters 12, 13]. And for those preferring Dutch: [Be, Chapter 11], [Ke, Chapter 12].

In this chapter the following topics will be treated:

- quadratic residues,
- the Legendre symbol,
- the Quadratic Reciprocity Law,
- the Jacobi symbol,
- modular square roots.

4.1 Quadratic residues and the Legendre symbol

In many applications, also in cryptography, it is useful to know which elements of $\mathbb{Z}_n^*$ can be squares, and which cannot.

Let $a \in \mathbb{Z}_n^*$. Then we say that a is a *quadratic residue* $\pmod n$ if there exists a $b \in \mathbb{Z}_n^*$ such that $a \equiv b^2 \pmod n$. If such b does not exist, then we say that a is a *quadratic nonresidue* $\pmod n$.

Example: the quadratic residues modulo 11 are $1, 3, 4, 5, 9$, and the quadratic nonresidues are $2, 6, 7, 8, 10$.

When n is not prime, then a quadratic residue modulo n necessarily is a quadratic residue modulo any prime divisor of n. We therefore proceed with studying quadratic residues modulo an odd prime p (even primes are not that interesting), and as quadratic residues modulo composite moduli turn out to be not too useful, we will further neglect them.

Lemma 4.1 *Let p be an odd prime.*
(a) *There are exactly $\frac{1}{2}(p-1)$ quadratic residues, and $\frac{1}{2}(p-1)$ quadratic nonresidues* $\pmod p$.
(b) *Let a be such that $p \nmid a$. Then a is a quadratic residue if and only if $a^{\frac{1}{2}(p-1)} \equiv 1 \pmod p$, and a is a quadratic nonresidue if and only if $a^{\frac{1}{2}(p-1)} \equiv -1 \pmod p$.*

Proof. **(a)** Let g be a primitive root $\pmod p$. Then: g^a is a quadratic residue $\pmod p$ if and only if there is a b such that $g^a \equiv \left(g^b\right)^2 = g^{2b} \pmod p$, if and only if $a \equiv 2b \pmod{p-1}$ (Exercise 3.4), if and only if a is even (because $p-1$ is even). Clearly the quadratic residues are precisely $1, g^2, g^4, \ldots, g^{p-3}$.
An alternative proof: Let g be a primitive root $\pmod p$, so $\mathbb{Z}_p^* = \left\{1, g, g^2, \ldots, g^{p-2}\right\}$. Any quadratic residue a satisfies $a \equiv b^2 \pmod p$ for some b, so by Fermat, $a^{(p-1)/2} \equiv b^{p-1} \equiv 1 \pmod p$. Theorem 3.7 says that there are precisely $\frac{1}{2}(p-1)$ roots of $x^{(p-1)/2} - 1$. Clearly $1, g^2, g^4, \ldots, g^{p-3}$ are $\frac{1}{2}(p-1)$ different quadratic residues, so these are all. Hence $g, g^3, \ldots, g^{p-2}$ are the quadratic nonresidues, and their number is $\frac{1}{2}(p-1)$.
(b) Let $c = a^{\frac{1}{2}(p-1)} \pmod p$. By Fermat's Theorem 3.3, c satisfies $c^2 = a^{p-1} \equiv 1 \pmod p$. It follows that $p \mid c^2 - 1 = (c-1)(c+1)$, and because p is prime, $c \equiv \pm 1 \pmod p$.
Let g be a primitive root $\pmod p$. There exists a $y \in \mathbb{N}$ such that $a \equiv g^y \pmod p$. Hence $c \equiv g^{\frac{1}{2}(p-1)y} \equiv 1 \pmod p$, if and only if $\mathrm{ord}(g) \mid \frac{1}{2}(p-1)y$, if and only if $p-1 \mid \frac{1}{2}(p-1)y$, if and only if y is even, if and only if a is a quadratic residue. □

In particular, the above lemma shows that -1 is a quadratic residue modulo p if $p \equiv 1 \pmod 4$, and -1 is a quadratic nonresidue modulo p if $p \equiv 3 \pmod 4$.

A convenient shorthand notation expressing the quadratic residuosity of a number a modulo a prime p is the *Legendre symbol*, defined (for all $a \in \mathbb{Z}$) as

$$\left(\frac{a}{p}\right) = \begin{cases} 1 & \text{if } a \text{ is a quadratic residue } \pmod p, \\ -1 & \text{if } a \text{ is a quadratic nonresidue } \pmod p, \\ 0 & \text{if } p \mid a. \end{cases}$$

Elementary properties of the Legendre symbol are given in the next result.

Lemma 4.2 *Let p be an odd prime.*
(a) $\left(\frac{a}{p}\right) \equiv a^{\frac{1}{2}(p-1)} \pmod p$.
(b) $\left(\frac{a}{p}\right) = \left(\frac{b}{p}\right)$ *if* $a \equiv b \pmod p$.
(c) $\left(\frac{ab}{p}\right) = \left(\frac{a}{p}\right)\left(\frac{b}{p}\right)$.
(d) *If* $p \nmid a$ *then* $\left(\frac{a^2}{p}\right) = 1$.

Proof. (a) is Lemma 4.1(b), (b) is trivial, (c) follows at once from (a), and (d) is trivial. □

4.2 The Quadratic Reciprocity Law

A less elementary but also very useful property of the Legendre symbol is the so called *Quadratic Reciprocity Law*, which we will prove below (it is kind of a sport for number theorists to come up with new proofs; more than 200 different proofs are known[1]; we can safely take the risk to believe that the law is true).

Theorem 4.3 (Quadratic Reciprocity Law) *Let p, q be distinct odd primes. If $p \equiv q \equiv 3 \pmod 4$ then $\left(\frac{p}{q}\right) = -\left(\frac{q}{p}\right)$, otherwise $\left(\frac{p}{q}\right) = \left(\frac{q}{p}\right)$.*

[1] See `http://www.rzuser.uni-heidelberg.de/~hb3/fchrono.html`.

Important special values are given in the next lemma.

Lemma 4.4 *Let p be an odd prime.*
(a) $\left(\frac{-1}{p}\right) = 1$ *if* $p \equiv 1 \pmod 4$, $\left(\frac{-1}{p}\right) = -1$ *if* $p \equiv 3 \pmod 4$.
(b) $\left(\frac{2}{p}\right) = 1$ *if* $p \equiv 1, 7 \pmod 8$, $\left(\frac{2}{p}\right) = -1$ *if* $p \equiv 3, 5 \pmod 8$.

The remainder of this section is devoted to proving these results. We start with an auxiliary lemma due to Gauss.

Let $a \in \mathbb{Z}_p^*$, for an odd prime p. We look at the set $a, 2a, 3a, \ldots, \frac{p-1}{2}a$, and view them as integers $\pmod p$, where we take representatives in the set $\left\{-\frac{p-1}{2}, -\frac{p-3}{2}, \ldots, -1, 0, 1, \ldots, \frac{p-3}{2}, \frac{p-1}{2}\right\}$. Those in $\left\{-\frac{p-1}{2}, \ldots, -1\right\}$ are called *negative representatives*, and those in $\left\{1, \ldots, \frac{p-1}{2}\right\}$ are called *positive representatives*.

The numbers $\pm a, \pm 2a, \ldots, \pm\frac{p-1}{2}a$ are $\pmod p$ all nonzero and all different. Hence $\left\{\pm a, \pm 2a, \ldots, \pm\frac{p-1}{2}a\right\} = \left\{\pm 1, \pm 2, \ldots, \pm\frac{p-1}{2}\right\} \pmod p$. For each $s = 1, 2, \ldots, \frac{p-1}{2}$ the pair $\pm sa$ has as representatives numbers $\pm u_s$, where for u_s we take the positive representative of the pair, i.e. $u_s \in \left\{1, 2, \ldots, \frac{p-1}{2}\right\}$. The observation that $\left\{u_1, u_2, \ldots, u_{(p-1)/2}\right\} = \left\{1, 2, \ldots, \frac{p-1}{2}\right\}$, is used below.

Lemma 4.5 (Gauss' Lemma) *Let $a \in \mathbb{Z}_p^*$, for an odd prime p, and let k_a be the number of $a, 2a, \ldots, \frac{p-1}{2}a$ that have negative representatives. Then $\left(\frac{a}{p}\right) = (-1)^{k_a}$.*

Proof. From each pair $\pm u_s$ one is the representative of sa, and its sign tells us whether it is a positive or negative representative. Multiplying all these representatives and using the above observation, we find

$$\left(\frac{p-1}{2}\right)!\, a^{\frac{p-1}{2}} = \prod_{s=1}^{\frac{p-1}{2}} sa \equiv \prod_{s=1}^{\frac{p-1}{2}} \pm u_s = (-1)^{k_a} \left(\frac{p-1}{2}\right)! \pmod p,$$

and the result follows at once by Lemma 4.1(b). □

We can now find the Legendre symbols $\left(\frac{-1}{p}\right)$ and $\left(\frac{2}{p}\right)$.

Proof of Lemma 4.4. (a) follows from Lemma 4.2(a). To prove (b) we have, by Gauss' Lemma 4.5, to count the number of $2, 4, \ldots, p-1$ that have negative representatives. This is just the number k_2 of even numbers between $\frac{1}{2}p$ and p. They are:

$$\begin{array}{lcll} p = 8\ell + 1 & : & 4\ell + 2, \ldots, 8\ell & \text{so} \quad k_2 = 2\ell, \\ p = 8\ell + 3 & : & 4\ell + 2, \ldots, 8\ell + 2 & \text{so} \quad k_2 = 2\ell + 1, \\ p = 8\ell + 5 & : & 4\ell + 4, \ldots, 8\ell + 4 & \text{so} \quad k_2 = 2\ell + 1, \\ p = 8\ell + 7 & : & 4\ell + 4, \ldots, 8\ell + 6 & \text{so} \quad k_2 = 2\ell + 2. \end{array}$$

The result now follows by Lemma 4.1(b). □

To prove the Quadratic Reciprocity Law, we follow an argument by Eisenstein. Note that in Gauss' Lemma we are interested in k_a (the number of negative representatives) only (mod 2). And note that if sa has a positive representative u_s then $sa = \left\lfloor \frac{sa}{p} \right\rfloor p + u_s$, whereas if sa has a negative representative $-u_s$ then $sa = \left\lfloor \frac{sa}{p} \right\rfloor p + p - u_s$. It seems that adding all sa's may yield information about k_a, and that's why we introduce

$$S(a,p) = \sum_{s=1}^{\frac{p-1}{2}} \left\lfloor \frac{sa}{p} \right\rfloor .$$

Let p, q be distinct odd prime numbers. We apply the above with $a = q$.

Lemma 4.6 (Eisenstein's First Lemma)

$$S(q,p) \equiv k_q \pmod 2.$$

Proof. The observation just above Gauss' Lemma 4.5 says that $\sum_{s=1}^{\frac{p-1}{2}} s = \sum_{s=1}^{\frac{p-1}{2}} u_s$. Now we have

$$q \sum_{s=1}^{\frac{p-1}{2}} s = \sum_{s=1}^{\frac{p-1}{2}} sq = \sum_{s=1}^{\frac{p-1}{2}} \left\lfloor \frac{sq}{p} \right\rfloor p + k_q p + \sum_{s=1}^{\frac{p-1}{2}} \pm u_s = p(S(q,p) + k_q) + \sum_{s=1}^{\frac{p-1}{2}} \pm u_s.$$

Modulo 2 we have $p \equiv 1$, $q \equiv 1$, and $\pm 1 \equiv 1$, so we get

$$S(q,p) + k_q \equiv \sum_{s=1}^{\frac{p-1}{2}} s - \sum_{s=1}^{\frac{p-1}{2}} u_s = 0 \pmod 2,$$

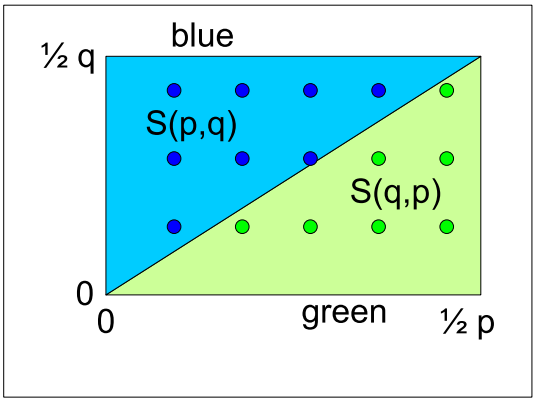
and we're done. □

Note that together with Gauss' Lemma we now have $\left(\frac{q}{p}\right) = (-1)^{S(q,p)}$.

Lemma 4.7 (Eisenstein's Second Lemma)

$$S(p,q) + S(q,p) = \frac{p-1}{2} \cdot \frac{q-1}{2}.$$

Proof of Lemma 4.7. The figure below shows a rectangle with vertices $(0,0), (\frac{1}{2}p, 0), (0, \frac{1}{2}q), (\frac{1}{2}p, \frac{1}{2}q)$, in which we count *lattice points*, i.e. points with integral coordinates. There clearly are $\frac{p-1}{2} \cdot \frac{q-1}{2}$ such points.
The rectangle is divided into two triangles by the diagonal given by $py = qx$. We will count the lattice points inside the two triangles, and show that they equal $S(p,q)$, resp. $S(q,p)$.



Note that there are no lattice points on the edges of the triangles (i.e. also not on the diagonal). The part of the sth row inside the upper left blue triangle is given by $y = s$ and $1 \leq x < \frac{p}{q}s$. The number of lattice points on the sth row inside the blue triangle therefore is $\left\lfloor \frac{sp}{q} \right\rfloor$. So inside the blue triangle there are exactly $\sum_{s=1}^{\frac{q-1}{2}} \left\lfloor \frac{sp}{q} \right\rfloor$, i.e. $S(p,q)$ points.

The part of the sth column inside the lower right green triangle is given by $x = s$ and $1 \leq y < \frac{q}{p}s$. The number of lattice points on the sth column inside the green triangle therefore is $\left\lfloor \frac{sq}{p} \right\rfloor$. So inside the green triangle there are exactly $\sum_{s=1}^{\frac{p-1}{2}} \left\lfloor \frac{sq}{p} \right\rfloor$, i.e. $S(q,p)$ points. □

Proof of Theorem 4.3. Combining Gauss' Lemma 4.5 with Eisenstein's Lemmas 4.6 and 4.7 we see that

$$\left(\frac{p}{q}\right)\left(\frac{q}{p}\right) = (-1)^{S(p,q)+S(q,p)} = (-1)^{\frac{p-1}{2}\cdot\frac{q-1}{2}},$$

and this suffices. □

4.3 Another proof

As said there are many proofs of the Quadratic Reciprocity Law. Here is another neat one, discovered in 2008 by Wouter Castryck from Leuven.

Let p, q be distinct odd primes. For odd $n \in \mathbb{N}$, let N_n be the number of solutions $(x_1, \ldots, x_n) \in \mathbb{Z}_q^n$ of

$$x_1^2 - x_2^2 + x_3^2 - \cdots + x_n^2 \equiv 1 \pmod{q}. \tag{4.1}$$

The idea is to count N_n in two different ways.

The number of solutions of (4.1) with $x_1 \equiv x_2 \pmod{q}$ is qN_{n-2}, because there are q possibilities for $x_1 (\equiv x_2 \pmod{q})$ and N_{n-2} for $(x_3, \ldots, x_n)$. To count the number of solutions of (4.1) with $x_1 \not\equiv x_2 \pmod{q}$, note that there are q^{n-2} possibilities for $(x_3, \ldots, x_n)$, and for any c (in fact, $c = 1 - x_3^2 + \ldots - x_n^2$) we count the number of solutions of $x_1^2 - x_2^2 \equiv c \pmod{q}$. By writing $y = x_1 - x_2 \not\equiv 0 \pmod{q}$ and $z = x_1 + x_2$ (the mapping from (x_1, x_2) to (y, z) is one to one) we see that each possible y leads to one solution: $z \equiv cy^{-1} \pmod{q}$, so there are $q - 1$ solutions of $x_1^2 - x_2^2 \equiv c \pmod{q}$. Hence the number of solutions of (4.1) with $x_1 \not\equiv x_2 \pmod{q}$ is $q^{n-2}(q-1)$, and for the total number of solutions of (4.1) we thus find $N_n = qN_{n-2} + q^{n-2}(q-1)$. So

$$N_n - q^{n-1} = q\left(N_{n-2} - q^{n-3}\right) = q^2\left(N_{n-4} - q^{n-5}\right) = \ldots = q^{(n-1)/2}\left(N_1 - 1\right),$$

and with $N_1 = 2$ this proves the expression

$$N_n = q^{n-1} + q^{(n-1)/2}. \tag{4.2}$$

Another way of counting N_n is as follows. The number of solutions $x \in \mathbb{Z}_q$ of $x^2 \equiv t \pmod q$ is equal to $1 + \left(\frac{t}{q}\right)$. So

$$\begin{aligned} N_n &= \sum_{\substack{t_1,\ldots,t_n \in \mathbb{Z}_q \\ t_1+\ldots+t_n \equiv 1 \\ (\mathrm{mod}\ q)}} \#\left\{x_1 \in \mathbb{Z}_q \mid x_1^2 \equiv t_1 \pmod q\right\} \#\left\{x_2 \in \mathbb{Z}_q \mid x_2^2 \equiv -t_2 \pmod q\right\} \cdots \\ &\qquad \cdots \#\left\{x_n \in \mathbb{Z}_q \mid x_n^2 \equiv t_n \pmod q\right\} \\ &= \sum_{\substack{t_1,\ldots,t_n \in \mathbb{Z}_q \\ t_1+\ldots+t_n \equiv 1 \\ (\mathrm{mod}\ q)}} \left(1 + \left(\frac{t_1}{q}\right)\right)\left(1 + \left(\frac{-t_2}{q}\right)\right) \ldots \left(1 + \left(\frac{t_n}{q}\right)\right). \end{aligned}$$

In the expanded product, all terms, except the first and last, are of the type

$$\sum_{\substack{t_1,\ldots,t_n \in \mathbb{Z}_q \\ t_1+\ldots+t_n \equiv 1 \\ (\mathrm{mod}\ q)}} \left(\frac{\pm t_{i_1}}{q}\right)\left(\frac{\pm t_{i_2}}{q}\right)\cdots\left(\frac{\pm t_{i_r}}{q}\right) = \pm q^{n-1-r} \sum_{t_{i_1},\ldots,t_{i_r} \in \mathbb{Z}_q} \left(\frac{t_{i_1}}{q}\right)\left(\frac{t_{i_2}}{q}\right)\cdots\left(\frac{t_{i_r}}{q}\right)$$

with $0 < r < n$. Because $\frac{1}{2}(q-1)$ of the $\left(\frac{t}{q}\right)$ for $t \in \mathbb{Z}_q$ are $+1$, an equal number are -1, and one is 0, we have $\sum_{t \in \mathbb{Z}_q} \left(\frac{t}{q}\right) = 0$, so all the terms above vanish:

$$\sum_{t_{i_1},\ldots,t_{i_r} \in \mathbb{Z}_q} \left(\frac{t_{i_1}}{q}\right)\cdots\left(\frac{t_{i_r}}{q}\right) = \left(\sum_{t_{i_1} \in \mathbb{Z}_q} \left(\frac{t_{i_1}}{q}\right)\right)\cdots\left(\sum_{t_{i_r} \in \mathbb{Z}_q} \left(\frac{t_{i_r}}{q}\right)\right) = 0.$$

From the expanded product only the first and last terms are left. They respectively are $\sum_{\substack{t_1,\ldots,t_n \in \mathbb{Z}_q \\ t_1+\ldots+t_n \equiv 1 \\ (\mathrm{mod}\ q)}} 1 = q^{n-1}$ and $\sum_{\substack{t_1,\ldots,t_n \in \mathbb{Z}_q \\ t_1+\ldots+t_n \equiv 1 \\ (\mathrm{mod}\ q)}} \left(\frac{t_1(-t_2)\cdots t_n}{q}\right)$. Using $\left(\frac{-1}{q}\right) = (-1)^{(q-1)/2}$ we thus find

$$N_n = q^{n-1} + (-1)^{(q-1)(n-1)/4} \sum_{\substack{t_1,\ldots,t_n \in \mathbb{Z}_q \\ t_1+\ldots+t_n \equiv 1 \\ (\mathrm{mod}\ q)}} \left(\frac{t_1 t_2 \cdots t_n}{q}\right). \tag{4.3}$$

Expressions (4.2) and (4.3) imply that

$$\sum_{\substack{t_1,\ldots,t_n \in \mathbb{Z}_q \\ t_1+\ldots+t_n \equiv 1 \\ (\mathrm{mod}\ q)}} \left(\frac{t_1 t_2 \cdots t_n}{q}\right) = (-1)^{(n-1)(q-1)/4} q^{(n-1)/2}.$$

Now we take $n = p$ prime. If $t_1 \equiv t_2 \equiv \ldots \equiv t_p \pmod p$ then the condition $t_1 + \ldots + t_p \equiv 1 \pmod q$ implies $t_1 \equiv \ldots \equiv t_p \equiv p^{-1} \pmod q$. By taking cyclic shifts, all other $(t_1, t_2, \ldots, t_p)$ fall

into disjunct sets of p elements each, such that all elements in one set have identical $t_1 t_2 \cdots t_p$ (note that this is in accordance with Fermat's theorem $q^{p-1} \equiv 1 \pmod{p}$). So taking the sum (mod p) all the contributions of those sets vanish, and we arrive at

$$\left(\frac{p^{-p}}{q}\right) \equiv (-1)^{(p-1)(q-1)/4} q^{(p-1)/2} \pmod{p}.$$

The Quadratic Reciprocity Law follows by noting that $\left(\frac{p^{-p}}{q}\right) = \left(\frac{p}{q}\right)$, that $q^{(p-1)/2} \equiv \left(\frac{q}{p}\right)$ (mod p), and that $(-1)^{(p-1)(q-1)/4}$ is just a cumbersome way of writing 1 if $p \equiv 1 \pmod 4$ or $q \equiv 1 \pmod 4$, and -1 if $p \equiv q \equiv 3 \pmod 4$.

4.4 The Jacobi symbol

With Lemma 4.2, Theorem 4.3 and Lemma 4.4 at our disposal we have a number of powerful tools to efficiently compute Legendre symbols, without having to do the expensive modular exponentiation $a^{\frac{1}{2}(p-1)} \pmod{p}$. The idea is to reduce a modulo p whenever $a > p$, to factor a completely into its prime divisors, and to use quadratic reciprocity to move to smaller primes in the 'denominator'.

For example, say we want to know whether 70 is a quadratic residue modulo 107. Then we factor 70, and by the quadratic reciprocity law we have $\left(\frac{70}{107}\right) = \left(\frac{2}{107}\right)\left(\frac{5}{107}\right)\left(\frac{7}{107}\right) = (-1)\left(\frac{107}{5}\right)(-\left(\frac{107}{7}\right))$. Now we reduce 107 (mod 5) and (mod 7), and we get $\left(\frac{70}{107}\right) = \left(\frac{2}{5}\right)\left(\frac{2}{7}\right) = (-1)1 = -1$. Hence 70 is a quadratic nonresidue modulo 107.

The main problem with the above method is that factoring is usually difficult. Therefore the Legendre symbol has been generalised to the so called Jacobi symbol $\left(\frac{a}{n}\right)$, in which n is not necessarily prime anymore. This is more convenient to compute.

Indeed, for integers m, n with $n \geq 3$ odd we define the *Jacobi symbol* $\left(\frac{m}{n}\right)$ in such a way that it has useful multiplicative properties. Indeed, when the prime factorization of n is given by $n = \prod_{i=1}^{k} p_i^{e_i}$, then we define

$$\left(\frac{m}{n}\right) = \prod_{i=1}^{k} \left(\frac{m}{p_i}\right)^{e_i},$$

where $\left(\frac{m}{p_i}\right)$ is the Legendre symbol.

You should however notice that the Jacobi symbol $\left(\frac{m}{n}\right)$ has no direct relation anymore to m being a quadratic residue or nonresidue modulo n (see Exercise 4.2). The only reason to introduce it is that it eases the computation of Legendre symbols.

The Jacobi symbol has the following properties.

Lemma 4.8 *Let k, n be odd integers ≥ 3, and let m be any integer.*
(a) *If n is prime then the Jacobi symbol $\left(\frac{m}{n}\right)$ equals the Legendre symbol $\left(\frac{m}{n}\right)$.*

(b) *If* $\gcd(m,n) \neq 1$ *then* $\left(\frac{m}{n}\right) = 0$. *And if* $\gcd(m,n) = 1$ *then* $\left(\frac{m}{n}\right) = \pm 1$.
(c) *If* $\gcd(m,n) = 1$ *then* $\left(\frac{m^2}{n}\right) = 1$.
(d) $\left(\frac{m}{n}\right) = \left(\frac{\ell}{n}\right)$ *if* $m \equiv \ell \pmod{n}$.
(e) $\left(\frac{m\ell}{n}\right) = \left(\frac{m}{n}\right)\left(\frac{\ell}{n}\right)$.
(f) $\left(\frac{m}{kn}\right) = \left(\frac{m}{k}\right)\left(\frac{m}{n}\right)$.
(g) **[Quadratic Reciprocity Law]**
If $k \equiv n \equiv 3 \pmod 4$ *then* $\left(\frac{k}{n}\right) = -\left(\frac{n}{k}\right)$, *otherwise* $\left(\frac{k}{n}\right) = \left(\frac{n}{k}\right)$.
(h) $\left(\frac{-1}{n}\right) = 1$ *if* $n \equiv 1 \pmod 4$, $\left(\frac{-1}{n}\right) = -1$ *if* $n \equiv 3 \pmod 4$.
(i) $\left(\frac{2}{n}\right) = 1$ *if* $n \equiv 1, 7 \pmod 8$, $\left(\frac{2}{n}\right) = -1$ *if* $n \equiv 3, 5 \pmod 8$.

Proof. Parts (a) to (f) follow easily from the corresponding properties of the Legendre symbol.
For (g), let n have the prime factorization $n = \prod_{i=1}^{r} p_i^{e_i}$, and let k have the prime factorization $k = \prod_{j=1}^{s} q_j^{f_j}$. Without loss of generality we may assume that the set of the p_i is disjunct from the set of the q_j, ensuring that all Legendre and Jacobi symbols below are nonzero.
Let $N = \#\{i \mid 1 \leq i \leq r \text{ and } p_i \equiv 3 \pmod 4 \text{ and } e_i \text{ is odd}\}$, and similarly let $K = \#\{j \mid 1 \leq j \leq s \text{ and } q_j \equiv 3 \pmod 4 \text{ and } f_j \text{ is odd}\}$. It easily follows that $n \equiv 1 \pmod 4$ if N is even, and $n \equiv 3 \pmod 4$ if N is odd, and similarly for k and K.
The quadratic reciprocity law for Legendre symbols shows that $\left(\left(\frac{p_i}{q_j}\right)\left(\frac{q_j}{p_i}\right)\right)^{e_i f_j} = -1$ if and only if $p_i \equiv q_j \equiv 3 \pmod 4$ and both e_i, f_j are odd. Now (e) and (f) imply

$$\left(\frac{n}{k}\right)\left(\frac{k}{n}\right) = \prod_{i=1}^{r}\prod_{j=1}^{s}\left(\left(\frac{p_i}{q_j}\right)\left(\frac{q_j}{p_i}\right)\right)^{e_i f_j} = (-1)^{NK},$$

which equals -1 if and only if both N and K are odd, which happens if and only if $n \equiv k \equiv 3 \pmod 4$.
Part (h) and (i) are left as an exercise (Exercise 4.6). □

Now we can use the above properties to compute the Jacobi (and hence the Legendre) symbol without having to factor numbers (other than splitting off factors 2, which is easy). For example, let us redo the computation of $\left(\frac{70}{107}\right)$. This is now done as follows: $\left(\frac{70}{107}\right) = \left(\frac{2}{107}\right)\left(\frac{35}{107}\right) = (-1)(-\left(\frac{107}{35}\right)) = \left(\frac{2}{35}\right) = -1$. Note that we did not have to factor 35.

The following algorithm now should be clear. It can be used to compute Legendre symbols by using Jacobi symbols.

Algorithm 4.1 (Jacobi Symbol)

Input: $m, n \in \mathbb{Z}$ `with` $n \geq 3$ `odd`
Output: $\left(\frac{m}{n}\right)$

Step 1: $m' \leftarrow m \pmod{n}$, $n' \leftarrow n$, $j \leftarrow 1$
Step 2: `while` $m' \neq 0$ `and` $n' > 1$ `do`
  `while` m' `is even do`
    $m' \leftarrow m'/2$
    `if` $n' \equiv 3$ `or` $5 \pmod 8$ `then` $j \leftarrow -j$
  $(m', n') \leftarrow (n', m')$
  `if` $m' \equiv n' \equiv 3 \pmod 4$ `then` $j \leftarrow -j$
  `if` $n' > 1$ `then` $m' \leftarrow m' \pmod{n'}$
Step 3: `if` $m' = 0$ `then` $j \leftarrow 0$
`output` j

The complexity of this algorithm is comparable to the Euclidean Algorithm, which should not come as a surprise, since the algorithm has a somewhat similar structure: at each step one number is reduced modulo the other, and then they are swapped. The bit complexity is easily seen to be $O((\log m)(\log n))$.

To avoid long division in the reduction step it is also possible to use a binary variant of the above algorithm.

Algorithm 4.2 (Jacobi Symbol, binary method)

Input: $m, n \in \mathbb{Z}$ `with` $n \geq 3$ `odd`
Output: $\left(\frac{m}{n}\right)$

Step 1: $m' \leftarrow |m|$, $n' \leftarrow n$
`if` $m < 0$ `and` $n \equiv 3 \pmod 4$ `then` $j \leftarrow -1$ `else` $j \leftarrow 1$
`while` $m' > 0$ `and` m' `is even do`
  $m' \leftarrow m'/2$
  `if` $n' \equiv 3$ `or` $5 \pmod 8$ `then` $j \leftarrow -j$
Step 2: `while` $m' \neq 0$ `and` $n' > 1$ `do`
  `while` $m' \geq n'$ `do`
    $m' \leftarrow m' - n'$
    `while` $m' > 0$ `and` m' `is even do`
      $m' \leftarrow m'/2$
      `if` $n' \equiv 3$ `or` $5 \pmod 8$ `then` $j \leftarrow -j$
  $(m', n') \leftarrow (n', m')$
  `if` $m' \equiv n' \equiv 3 \pmod 4$ `then` $j \leftarrow -j$
Step 3: `if` $m' = 0$ `then` $j \leftarrow 0$
`output` j

To conclude this section we discuss the problem of finding numbers that are quadratic residues or quadratic nonresidues.

To find for given prime modulus p a (random) quadratic residue is trivial: take a random number and square it. To find a (random) quadratic nonresidue is easy when $p \equiv 3 \pmod 4$ or $p \equiv 5 \pmod 8$: simply take a random quadratic residue and multiply it by -1 resp. 2.

Given the above efficient algorithms, generating a quadratic nonresidue modulo p when $p \equiv 1 \pmod 8$ is in practice possible by trial and error: simply take a random element and compute the Legendre symbol, repeat this until the Legendre symbol is -1. On average one should succeed after 2 trials.

4.5 Modular Square Roots

With the Legendre symbol and the algorithm for computing it we have a way to find out whether a given number a is a quadratic residue $\pmod p$ or not, but we do not yet have a method of finding the *modular square root* b such that $b^2 \equiv a \pmod p$.

For an odd prime modulus p we now give the method of computing the square roots of a, i.e. of computing b such that $b^2 \equiv a \pmod p$. Note that if a is a nonzero quadratic residue modulo a prime p, then there are 2 square roots $\pmod p$, since $x^2 \equiv b^2 \pmod p$ implies $p \mid x^2 - b^2 = (x-b)(x+b)$, so $p \mid x-b$ or $p \mid x+b$, so the square roots are $x \equiv \pm b \pmod p$.

First we treat the cases where $p \not\equiv 1 \pmod 8$, as these turn out to be easy.

Lemma 4.9 *Let p be an odd prime, and $a \not\equiv 0 \pmod p$ a quadratic residue $\pmod p$.*
(a) *If $p \equiv 3 \pmod 4$, then $b = a^{\frac{1}{4}(p+1)}$ is a square root of $a \pmod p$.*
(b) *If $p \equiv 5 \pmod 8$, then $a^{\frac{1}{4}(p-1)} \equiv \pm 1 \pmod p$.*
In the case $a^{\frac{1}{4}(p-1)} \equiv 1 \pmod p$ a square root of $a \pmod p$ is given by $b = a^{\frac{1}{8}(p+3)}$.
In the case $a^{\frac{1}{4}(p-1)} \equiv -1 \pmod p$ a square root of $a \pmod p$ is given by $b = 2^{\frac{1}{4}(p-1)} a^{\frac{1}{8}(p+3)}$.

Proof. See Exercise 4.7. □

If $p \equiv 1 \pmod 8$ then no deterministic method was known until 2006, when Christiaan van de Woestijne described such a method in his University of Leiden PhD thesis. Probabilistic methods have been known for more than a century, and are very practical. We give the algorithm due to Tonelli (that, by the way, works for all odd primes p).

Algorithm 4.3 (Square Root modulo a prime)

Input:	p `prime,`
	`a quadratic residue` a `modulo` p
Output:	b `such that` $b^2 \equiv a \pmod p$
Step 1:	`pick a random quadratic nonresidue` g
	`compute` s, t `such that` $p-1 = 2^s t$ `with` t `odd`
	$e \leftarrow 0$
Step 2:	`for` i `from 2 to` s `do`
	`if` $(ag^{-e})^{2^{s-i}t} \not\equiv 1 \pmod p$ `then` $e \leftarrow e + 2^{i-1}$
Step 3:	$h \leftarrow ag^{-e}$, $b \leftarrow g^{\frac{1}{2}e} h^{\frac{1}{2}(t+1)}$, `output` b

Lemma 4.10 *Let p be an odd prime, and $a \not\equiv 0 \pmod p$ a quadratic residue $\pmod p$. Then Algorithm 4.3 outputs a b such that $b^2 \equiv a \pmod p$.*

Proof. First note that at the beginning of Step 2 $ag^{-e} = a$, and $(ag^{-e})^{2^{s-1}t} = a^{(p-1)/2} \equiv 1 \pmod p$ as a is a quadratic residue. It follows that $(ag^{-e})^{2^{s-2}t} \equiv \pm 1 \pmod p$, and if it is -1, then e is increased by 2. This implies that $(ag^{-e})^{2^{s-2}t} \equiv (-1) g^{-2^{s-1}t} \pmod p$, and since g is a quadratic nonresidue, we have $g^{-2^{s-1}t} = g^{-(p-1)/2} \equiv -1 \pmod p$, so that after applying the inner loop in Step 2 for $i = 2$ we always have $(ag^{-e})^{2^{s-2}t} \equiv 1 \pmod p$. Now by induction (see Exercise 4.8) it follows that $(ag^{-e})^{2^{s-i}t} \equiv 1 \pmod p$ is always true after the inner loop in Step 2 has been done for some i. In particular, at the beginning of Step 3 e has been engineered such that $(ag^{-e})^t \equiv 1 \pmod p$. Since $b^2 = a^{t+1} g^{-et}$ it follows that $b^2 \equiv a \pmod p$. □

Note that this algorithm is probabilistic since no deterministic method is known to find a quadratic nonresidue.

For composite moduli of which the factorization is not known, the problem of finding square roots is believed to be very hard, i.e. an efficient method is not known. Even finding out whether or not a given number is a quadratic residue modulo such a composite modulus is not well understood in the case of the Jacobi symbol being 1.

When the factorization of n is known, say $n = \prod_{i=i}^{k} p_i^{e_i}$, the square roots modulo $p_i^{e_i}$ can be found, and then the Chinese Remainder Theorem can be applied to find all square roots modulo n. For each prime power there is only one pair of square roots of a, so the number of square roots modulo n is 2^k (unless some p_i divides a).

We do not work out the details, but for n being the product of distinct primes (i.e. $e_i = 1$ for all i) you should be able to do this.

Exercises

4.1. Prove that the product of two quadratic residues is a quadratic residue.
Prove that the product of two quadratic nonresidues is a quadratic residue.
Prove that the product of a quadratic residue and a quadratic nonresidue is a quadratic nonresidue.

4.2. Compute the Jacobi symbol $\left(\frac{8}{15}\right)$. Is 8 a quadratic residue modulo 15?

4.3. Compute the Jacobi symbol $\left(\frac{727}{1169}\right)$ by both the original and the binary algorithms.

4.4. Compute all odd primes p for which 6 is a quadratic residue $\pmod{p}$.

4.5. Compute all square roots of 51 modulo 91 (note: there are 4 of them).

4.6. Complete the proof of Lemma 4.8.

4.7. Prove Lemma 4.9.

4.8. Write out the induction argument in the proof of Lemma 4.10.

4.9. Write computer programs that compute Legendre and Jacobi symbols in an efficient way.

4.10. Let $n = pq$ for different odd primes p, q, and assume that you know n but you do not know p and q. Let a be a quadratic residue $\pmod{n}$. Show that if you have a way to find the four solutions b of $b^2 \equiv a \pmod{n}$, then you can compute p and q.

4.11. (a) Let p be an odd prime, and let $a \in \mathbb{Z}$ with $p \nmid a$ be a quadratic residue $\pmod{p}$. Let $b_1 \in \{1, 2, \ldots, p-1\}$ be such that $b_1^2 \equiv a \pmod{p}$. Let $k \geq 2$. Show that there exists a unique $b_k \in \{1, 2, \ldots, p^k - 1\}$ with $b_k \equiv b_1 \pmod{p}$ such that $b_k^2 \equiv a \pmod{p^k}$. Conclude that $x^2 \equiv a \pmod{p^k}$ has exactly 2 solutions.
Hint: show by induction that there exists a $b_k \in \{1, 2, \ldots, p^k - 1\}$ with $b_k \equiv b_{k-1} \pmod{p^{k-1}}$ such that $b_k^2 \equiv a \pmod{p^k}$.
(b) Let $n \geq 3$ be a positive odd integer, and let a be a quadratic residue $\pmod{p}$ with $\gcd(a, n) = 1$. Prove that there exist precisely 2^ℓ solutions of $x^2 \equiv a \pmod{n}$, where ℓ is the number of distinct prime factors of n.

Chapter 5

Prime Numbers

Introduction

General references for this chapter: [BS, Chapters 8, 9], [CP, Chapters 3, 4], [GG, Chapter 18], [Sh, Chapter 5, Section 7.5, Chapter 10, 22]. And for those preferring Dutch: [Be, Chapters 8, 19], [Ke, Chapter 13], [dW, Chapter 3].

In this chapter the following topics will be treated:

- prime number distribution,
- probabilistic primality tests,
- prime number generation.

5.1 Prime Number Distribution

5.1.1 The Prime Number Theorem

Prime numbers play an important role in discrete mathematics, especially in cryptography. In applications one often has to find large 'random' prime numbers (several thousands of bits). This means that it would be nice if number theory could guarantee that such large prime numbers exist in some abundance, and could provide efficient methods of generating them.

The first result in the theory of prime numbers is the fact that there are infinitely many.

Theorem 5.1 (Infinitude of primes) *There are infinitely many prime numbers.*

Proof. Let $p_1, p_2, \ldots, p_n$ be some set of primes. Then $P = p_1 p_2 \cdots p_n + 1$ is a number that is not divisible by any of $p_1, p_2, \ldots, p_n$. On the other hand, P has at least some prime factor p. Then p must be a prime different from $p_1, \ldots, p_n$. So for any finite set of primes there exists another prime that is not in that set. □

The main tool in studying the distribution of the primes is the *prime counting function*

$$\pi(x) = \#\{p \leq x | p \text{ prime}\}.$$

There is an extensive literature about $\pi(x)$, in an area of number theory called *analytic number theory*. This theory contains a large number of deep results, of which we can only touch the very surface, most of the time without proofs.

The main result from (analytic) prime number theory is the Prime Number Theorem, which estimates $\pi(x)$ in terms of known functions of x. To be able to state it, we need the concept of asymptotically equivalent functions. This concept describes what it means that two functions show the same growth behaviour for x tending towards infinity.

We say that $f(x)$ and $g(x)$ are *asymptotically equivalent as* $x \to \infty$, notation $f(x) \sim g(x)$, when their quotient tends to 1 as $x \to \infty$. In a formula:

$$f(x) \sim g(x) \quad \text{means} \quad \lim_{x\to\infty} \frac{f(x)}{g(x)} = 1.$$

Now we can state the *Prime Number Theorem*. Proving it however is way beyond the scope of this course.

Theorem 5.2 (Prime Number Theorem, Hadamard and de la Vallée-Poussin)

$$\pi(x) \sim \frac{x}{\log x}.$$

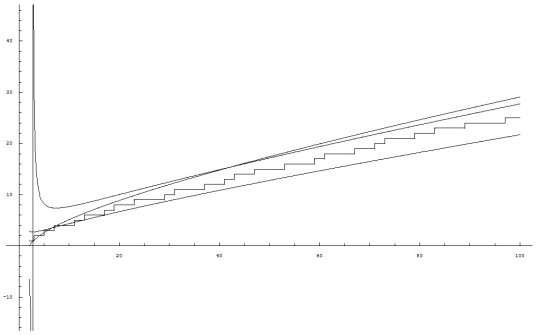
One way to look at this is in a probabilistic sense. Say we are looking for prime numbers in a certain interval $[x, x + \Delta x]$. For example, we might be looking for prime numbers of 1024 bits, then we have the interval $[2^{1023}, 2^{1024}]$. The Prime Number Theorem then says that the number of primes in this interval, $\pi(x + \Delta x) - \pi(x)$, is roughly $\dfrac{\Delta x}{\log x}$. As the total number of integers in the interval is Δx, the probability that a random number from the interval is prime is approximately $\dfrac{1}{\log x}$. As $\log 2^{1023} \approx 710$ this means that about one of every 710 numbers of 1024 bits is prime. This may look not very much, but note that all primes are odd, so we can immediately rule out all even numbers, and similarly we can e.g. rule out all multiples of 3. This leaves us with one out of every 237 numbers $\pm 1 \pmod 6$ having 1024 bits that is prime. Note that there are approximately $\frac{1}{710}2^{1023} \approx 10^{305}$ prime numbers in this interval. That is an abundance indeed (the number of elementary particles in the universe is estimated at only 10^{80}).

The estimate $\dfrac{x}{\log x}$ for $\pi(x)$ has a very simple form, the right asymptotic behaviour and is a reasonable approximation in practice. Nevertheless there are better estimates. One that is not very well known, but in practice almost as easy, and considerably better, is $\pi(x) \sim \dfrac{x}{\log x - 1}$. One also often sees the expression $\pi(x) \sim \mathrm{li}(x) = \displaystyle\int_2^x \frac{1}{\log t}\, dt$. This is an even closer approximation, but less practical.

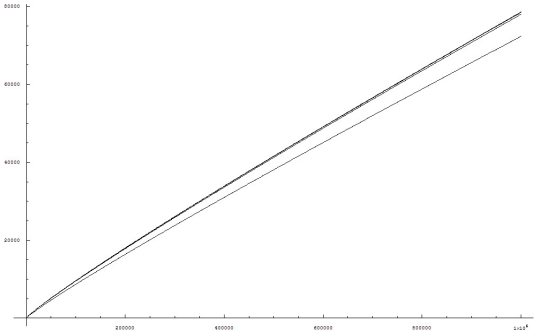
To get an idea of the accuracy of the estimates for $\pi(x)$, we show a small table and some graphs.

$\frac{\log x}{\log 10}$	$\pi(x)$	$\frac{x}{\log x}$	$\frac{x}{\log x-1}$	li(x)	$\pi(x)/\frac{x}{\log x}$	$\pi(x)/\frac{x}{\log x-1}$	li(x)	$\pi(x)-\frac{x}{\log x}$	$\pi(x)-\frac{x}{\log x-1}$	li(x)
1	4	4.3429	7.6770	5.1204	0.9210	0.5210	0.7812	−0.3429	−3.6770	−1.1204
2	25	21.715	27.738	29.081	1.1513	0.9013	0.8597	3.2853	−2.7379	−4.0810
3	168	144.77	169.27	176.56	1.1605	0.9925	0.9515	23.235	−1.2690	−8.5645
4	1229	1085.7	1218.0	1245.1	1.1320	1.0091	0.9871	143.26	11.024	−16.092
5	9592	8685.9	9512.1	9628.8	1.1043	1.0084	0.9962	906.11	79.900	−36.764
6	78498	72382.	78030.	78627.	1.0845	1.0060	0.9984	6115.6	467.55	−128.50
7	664579	6.204E5	6.615E5	6.649E5	1.0712	1.0047	0.9995	44158.	3120.0	−338.36
8	5761455	5.429E6	5.740E6	5.762E6	1.0613	1.0037	0.9999	3.328E5	21151.	−753.33
9	50847534	4.825E7	5.070E7	5.085E7	1.0537	1.0029	1.0000	2.593E6	1.460E5	−1699.9
10	455052511	4.343E8	4.540E8	4.551E8	1.0478	1.0023	1.0000	2.076E7	1.041E6	−3102.5

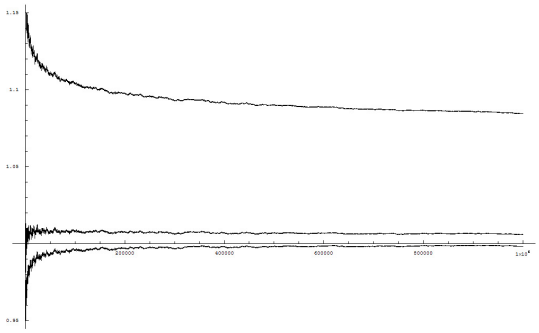
(notation: En stands for $\times 10^n$)



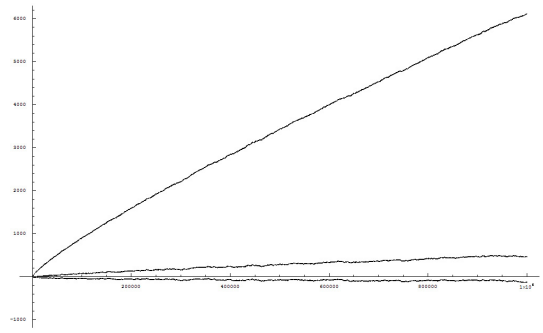
$\pi(x)$, $\dfrac{x}{\log x}$, $\dfrac{x}{\log x - 1}$ and $\mathrm{li}(x)$



$\pi(x)$, $\dfrac{x}{\log x}$, $\dfrac{x}{\log x - 1}$ and $\mathrm{li}(x)$



$\dfrac{\pi(x)}{\frac{x}{\log x}}$, $\dfrac{\pi(x)}{\frac{x}{\log x - 1}}$ and $\dfrac{\pi(x)}{\mathrm{li}(x)}$



$\pi(x) - \dfrac{x}{\log x}$, $\pi(x) - \dfrac{x}{\log x - 1}$ and $\pi(x) - \mathrm{li}(x)$

The graphs above suggest that there is a lot of regularity in the distribution of prime numbers, and that is to some extent true. The prime number distribution seems to have a lot of properties that are very plausible from an experimental point of view, but have not been proved. But there are also many irregularities.

The main conjecture in this area is the *Riemann Hypothesis*, a long standing open problem with a lot of consequences. The Riemann Hypothesis can be formulated as a very good estimate for the error term $\pi(x) - \mathrm{li}(x)$, as follows.

Conjecture 5.3 (Riemann Hypothesis) *For $x \geq 3$ we have*

$$|\pi(x) - \mathrm{li}(x)| < \sqrt{x} \log x.$$

Only the much weaker

Theorem 5.4 (Prime Number Theorem, error estimate) *For any $k > 0$*

$$\pi(x) = \mathrm{li}(x) + \mathrm{O}\left(\frac{x}{(\log x)^k}\right).$$

is known.

The table above gives some experimental evidence for Conjecture 5.3: $\pi(x)$ and $\mathrm{li}(x)$ coincide for about the first half of their digits.

The following lemma is also sometimes useful. It estimates the size of the n'th prime number.

Lemma 5.5 *Let p_n be the n'th prime number. Then $p_n \sim n \log n$.*

Also very explicit versions of the prime number theorem exist.

Theorem 5.6 (Prime Number Theorem, explicit version)

(a) $\frac{x}{2(\log x)^2} < \pi(x) - \frac{x}{\log x} < \frac{3x}{2(\log x)^2}$ *for* $x \geq 59$.

(b) $-\frac{1}{2}n < p_n - n(\log n + \log\log n - 1) < \frac{1}{2}n$ *for* $n \geq 20$.

There are gaps between consecutive primes as large as one wants, see Exercise 5.3. There is an extensive literature on the size of prime gaps. Bertrand's Postulate says that there is always a prime between n and $2n$. This has been improved to

Theorem 5.7 (Prime gaps, Baker and Harman) *Let $\epsilon > 0$, and let n be large enough. There always is a prime between n and $n + n^{0.535+\epsilon}$.*
Equivalently, $p_n - p_{n-1} \leq n^{0.535+\epsilon}$.

It is conjectured that the above theorem still is very pessimistic, namely that the right bound should be $p_n - p_{n-1} = \mathrm{O}((\log n)^2)$.

5.1.2 Probabilistic arguments

We conclude this section with a few lines about probabilistic (or heuristic) reasoning. Experimentally it usually appears to be true that different 'events' involving the primality of random numbers are independent, unless a good reason for dependence can be found. Some of the conjectures one can make as a result of such heuristic reasoning can be proved, while others remain unproven, though very probable.

An example of the first kind is the prime number theorem for arithmetic progressions. Let be given a modulus m, and a number $a \in \{0, 1, \ldots, m-1\}$. An *arithmetic progression* is a sequence like $\{a, a+m, a+2m, a+3m, \ldots\}$. We would like to know how many primes up to x are in such an arithmetic progression, i.e. in a certain congruence class modulo m. We then look at the events 'p is prime' and '$p \equiv a \pmod{m}$'.

If $\gcd(a, m) \neq 1$ then the events clearly are dependent, as a prime p can be congruent to a modulo m only if $\gcd(a, m) \mid p$, implying $p \mid a$ and $p \mid m$. Clearly there are only finitely many such primes p, and this means that for a random p the probability of the two events happening at the same time is 0.

But if a and m are coprime, the events seem to be independent. Heuristic reasoning now suggests that the primes are equally distributed over the congruence classes of a reduced residue system modulo any m. The number of primes p in an interval of length Δx around x is approximately $\frac{\Delta x}{\log x}$, so the mentioned heuristic means that for each a coprime to m the number of primes $p \equiv a \pmod{m}$ in that interval is approximately $\frac{1}{\phi(m)} \frac{\Delta x}{\log x}$.

This heuristic is indeed known to be true. Put

$$\pi_{a,m}(x) = \#\{p \leq x | p \text{ prime and } p \equiv a \pmod{m}\}.$$

Theorem 5.8 (Prime Number Theorem for Arithmetic Progressions, Dirichlet)
Let $m \in \mathbb{Z}$, $m \geq 2$, and $a \in \{0, 1, \ldots, m-1\}$ coprime to m. Then

$$\pi_{a,m}(x) \sim \frac{1}{\phi(m)} \frac{x}{\log x}.$$

A similar heuristic reasoning can be used to estimate the number of *twin primes*, i.e. the number of pairs $p-2, p$ below x that are both prime. As the events '$p-2$ is prime' and 'p is prime' can be assumed to be independent (when $p \equiv 1 \pmod 6$), the number of twin primes below x can be easily estimated to be $c\frac{x}{(\log x)^2}$ for a constant c. The correct value for c is the so called *twin prime constant* $c_2 = 2\prod_p \frac{p(p-2)}{(p-1)^2} = 1.3203\ldots$, where the product is taken over all odd primes. To obtain this value a more subtle probabilistic argument is required. A result like this however is not proven, it is not even known whether there are infinitely many twin primes. But experimentally the heuristic works very well.

Finally we mention *Sophie Germain primes*, which are primes p for which $\frac{1}{2}(p-1)$ also is prime. Note that when p is a Sophie Germain prime then the only possible orders of elements in $\mathbb{Z}_p^*$ are $1, 2, \frac{1}{2}(p-1)$ and $p-1$. For this reason Sophie Germain primes have gained some popularity in cryptographic applications, where they are also called *safe primes* or *strong primes*. Again assuming the heuristic that the events of p and $\frac{1}{2}(p-1)$ being prime are independent (then of course $p \equiv 3 \pmod 4$), the number of Sophie Germain primes below x can be estimated at $c\frac{x}{(\log x)^2}$ for a constant c. In fact, this time $c = \frac{1}{2}c_2 = 0.66016\ldots$, where c_2 is the twin prime constant.

5.2 Probabilistic Primality Testing

5.2.1 Introduction to Primality Testing

We now move to primality testing, i.e. given a (large) integer n, trying to answer the question "Is this integer n prime or composite?". The naive approach would be to list all prime numbers up to $\sqrt{n}$, and do division for each of them (why only to $\sqrt{n}$?). This is an exponential algorithm: the runtime is exponential in the length of the input. For n larger than 10^{15} or so this clearly is out of the question.

Primality tests come in two flavors. *Deterministic primality tests* give a Yes-or-No result: "Yes, n is proven to be prime", or "No, n is proven to be composite". *Probabilistic primality tests* are a bit weaker, as they give only a Probably-or-No result: "Probably n is prime", or "No, n is proven to be composite". The probability that the algorithm gives a wrong answer should be provably small. The main advantage of probabilistic tests is that they are much more efficient.

In practice (at least in cryptography) one usually is happy with a probabilistic test only. When a deterministic test is done, one usually first applies a probabilistic test as well, to quickly discard composite numbers. Also, to quickly discard composites that have a small prime divisor, some more naive methods can also be useful, such as first computing $\gcd(n, p_1p_2p_3\cdots p_k)$ for the smallest k primes (the value of the product of the first k primes can be precomputed and stored).

5.2.2 Pseudoprimes, Witnesses and Liars

The starting point for many probabilistic primality tests is Fermat's Theorem 3.3. It says that if p is prime and a some integer not divisible by p, then $a^{p-1} \equiv 1 \pmod p$. The fact that $2^{14} \equiv 4 \not\equiv 1 \pmod{15}$ thus proves that 15 is composite (without revealing any factors).

However, Fermat's Theorem cannot be used backwards. That is, for composite n there may exist integers a coprime to n satisfying $a^{n-1} \equiv 1 \pmod n$. For example, $4^{14} \equiv 1 \pmod{15}$.

Such a composite, that 'behaves like a prime' with respect to the base a, is called a *pseudoprime* or *Fermat pseudoprime* with respect to the base a. A number a coprime to n such that $a^{n-1} \not\equiv 1$

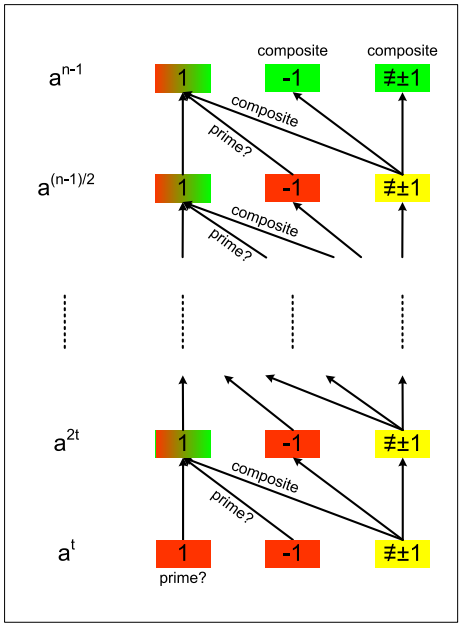
(mod n) is called a *Fermat witness* for the compositeness of n. A number a coprime to n such that $a^{n-1} \equiv 1 \pmod{n}$ while n is composite is called a *Fermat liar* for the primality of n.

So it may happen that a primality test based on Fermat's Theorem fails because one happens to have chosen a base a for which n is a pseudoprime. The situation is even worse: there exist numbers, the so called *Carmichael numbers*, that are pseudoprimes with respect to any base a. The smallest is 561 (see Exercise 5.6). Though Carmichael numbers seem rather sparse, actually there are quite a lot of them: the number of Carmichael numbers up to x is at least proportional to $x^{2/7}$.

5.2.3 The Miller-Rabin Test

Assume that we have a number n which we want to test for primality, and we have an integer a coprime to n, that satisfies $a^{n-1} \equiv 1 \pmod{n}$. So n is either prime or a pseudoprime with respect to a.

Another feature that distinguishes primes from composites is related to square roots. If n is prime, then there are only two square roots of 1 modulo n, namely ± 1. But if n is composite, there are more square roots of 1 modulo n. We know that $a^{n-1} \equiv 1 \pmod{n}$, and $n-1$ is even, and we can thus easily compute some square roots of a^{n-1}, namely $a^{\frac{1}{2}(n-1)} \pmod{n}$ (thus by modular exponentiation). In general, we start for some $r \in \mathbb{N}$ (at first $r = 1$) with an even $\frac{1}{2^{r-1}}(n-1)$, and explicitly known values of $a^{\frac{1}{2^r}(n-1)} \pmod{n}$. Now there are three possibilities.



1. If it happens that $a^{\frac{1}{2^r}(n-1)} \equiv -1 \pmod{n}$ then we stop because we cannot exclude that n is prime, and we don't know how to proceed with this a.
2. If it happens that $a^{\frac{1}{2^r}(n-1)} \not\equiv \pm 1 \pmod{n}$ then we have proved that n is composite.
3. If $a^{\frac{1}{2^r}(n-1)} \equiv 1 \pmod{n}$ then we increase r by 1, and if $\frac{1}{2^{r-1}}(n-1)$ (with the new r) is still even we compute the new $a^{\frac{1}{2^r}(n-1)} \pmod{n}$, and repeat the game; otherwise we stop.

In other words, we look at $a^{n-1}, a^{\frac{1}{2}(n-1)}, a^{\frac{1}{4}(n-1)}, \ldots, a^{\frac{1}{2^r}(n-1)} \pmod{n}$, until $\frac{1}{2^r}(n-1)$ has become odd (conclusion: n may be prime); or $a^{\frac{1}{2^r}(n-1)} \equiv -1 \pmod{n}$ (conclusion: n may be prime); or $a^{\frac{1}{2^r}(n-1)} \not\equiv \pm 1 \pmod{n}$ (conclusion: n is composite).

Again this is not yet a deterministic test: there are composite numbers n for which this test fails, the so called *strong pseudoprimes* with respect to the base a. Then a is called a *strong liar* for the primality of n. Similarly when the test succeeds in proving that n is composite, a is called a *strong witness* for the compositeness of n.

Example: $n = 561$ is a Carmichael number. Let us take $a = 206$, which by the Euclidean algorithm can be shown to be coprime to 561. We compute $206^{560} \equiv 1 \pmod{561}$ (no surprise, as 561 is Carmichael), then $206^{280} \equiv 1 \pmod{561}$, so we continue with $206^{140} \equiv 67 \pmod{561}$. The number 67 thus is a solution to $x^2 \equiv 1 \pmod{561}$ other than ± 1, and it follows that 561 is composite. So 206 is a strong witness for 561.

Example: $n = 13981 = 11 \times 31 \times 41$ has $a = 2$ as a strong liar. Namely, $2^{6990} \equiv -1 \pmod{13980}$.

In practice it is easiest to do the computation in the other direction: by repeated squaring in stead of repeated square rooting. The reason is that when $x \pmod n$ has been computed, then computing $x^2 \pmod n$ is easy, whereas the other way is difficult. So write $n-1 = 2^s t$ with t odd and $s \geq 1$. Then compute a^t, and by repeated squaring successively compute $a^{2t}, a^{4t}, \ldots, a^{2^{s-1}t} = a^{\frac{1}{2}(n-1)}, a^{2^s t} = a^{n-1} \pmod n$, until we hit $a^{2^k m} \equiv 1 \pmod n$. If the previous term $a^{2^{k-1}m} \pmod n$ is not -1, we have proved that n is composite.

This idea is turned into an algorithm below, due to Miller and Rabin, and thus known as the Miller-Rabin Primality Test. This algorithm accepts as input, next to the number n to be tested, a 'security parameter' R. This is the maximum number of different random a to be tested for being (Fermat or strong) liars.

We have the following result about the Miller-Rabin algorithm, which we cannot prove.

Theorem 5.9 (Miller-Rabin Primality Test)
(a) *If n is an odd prime then the Miller-Rabin algorithm outputs "probably prime".*
(b) *If n is an odd composite, then the Miller-Rabin algorithm with security parameter R outputs "composite" with probability $> 1 - \frac{1}{4^R}$, and "probably prime" with probability $\leq \frac{1}{4^R}$.*
(c) *The Miller-Rabin algorithm takes $\mathrm{O}(R(\log n)^3)$ bit operations.*

This result can be interpreted as follows, as the output phrases suggest:

Corollary 5.10
(a) *If on input n the Miller-Rabin algorithm outputs "composite", then n is composite.*
(b) *If on input n the Miller-Rabin algorithm with security parameter R outputs "probably prime", then n is prime with probability $> 1 - \frac{1}{4^R}$, and composite with probability $\leq \frac{1}{4^R}$.*

The probability bound appears experimentally to be far from sharp, i.e. in practice the Miller-Rabin primality test performs much better than Theorem 5.9(b) suggests. See [MvOV, Section 4.4.1] for a discussion.

Algorithm 5.1 (Miller-Rabin Primality Test)

Input: $n \in \mathbb{Z}$ `with` $n \geq 3$ `and odd`
$R \in \mathbb{N}$ `(security parameter)`
Output: `"probably prime" or "composite"`

Step 1: $r \leftarrow 0$, `prob-prime` $\leftarrow$ `true`
`compute` s, t `such that` $n - 1 = 2^s t$ `and` t `odd`
Step 2: `while prob-prime and` $r < R$ `do`
  Step 3: `choose a new random` $a \in \{2, 3, \ldots, n-2\}$
    $k \leftarrow 0$, $x \leftarrow a^t \pmod n$
  Step 4: `while` $k < s$ `and` $x \not\equiv 1 \pmod n$ `do`
    $k \leftarrow k + 1$, $z \leftarrow x$, $x \leftarrow x^2 \pmod n$
  Step 5: `if` $k = 0$
    `then` $r \leftarrow r + 1$
    `else`
      `if` $k = s$ `and` $x \not\equiv 1 \pmod n$
      `then prob-prime` $\leftarrow$ `false`
      `else`
        `if` $z \equiv -1 \pmod n$
        `then` $r \leftarrow r + 1$
        `else prob-prime` $\leftarrow$ `false`
Step 6: `if prob-prime`
`then output "probably prime"`
`else output "composite"`

5.3 Deterministic Primality Testing

5.3.1 The primitive root test

In this course we cannot go deep into the theory of deterministic primality tests. Some tests work very well for numbers of a special form. To give one example, consider the following lemma.

Lemma 5.11 *The number n is prime if and only if there exists an $a \in \mathbb{Z}_n^*$ that has order $n-1$.*

Recall that to find elements of order $n-1$ one has to check that $a^{n-1} \equiv 1 \pmod{n}$, and $a^{(n-1)/q} \not\equiv 1 \pmod{n}$ for all primes $q \mid n-1$, see Algorithm 3.1.

This lemma can be used in a primality test for n by choosing a number of random integers a and testing their orders. When n is prime then there are $\phi(n-1)$ primitive roots, and $\phi(n-1)$ cannot be small. In fact, $\phi(m) > c\dfrac{m}{\log\log m}$ for some constant c, so after at most $\mathrm{O}(\log\log n)$ trials one expects to have found a primitive root. If that happens, the number n is proven to be a prime.

When a primitive root is not found in $\mathrm{O}(\log\log n)$ trials, then one suspects that n might be composite, and the Miller-Rabin test is then likely to spot this.

The advantage of this idea is that if the algorithm returns with the decision "prime" (from a primitive root being found) or "composite" (from the Miller-Rabin test), then the number is a proven prime or composite. The algorithm is however not guaranteed to terminate.

In practice the most troublesome property of the algorithm is that the factorization of $n-1$ is required; this factorization is usually not available.

The idea has been developed further, e.g. such that only a partial factorization of $n-1$ is required, but we will not go into details.

5.3.2 Primality certificates

Some algorithms produce *primality certificates*. That is, for a given number n, after a long computation, a small set of small numbers is produced, with which a proof for the primality of n is easily verified. For example, a primitive root can act as a primality certificate.

Note that *compositeness certificates* are also possible, e.g. (Fermat or strong) witnesses, or a complete or partial factorization.

5.3.3 Other deterministic primality tests

For numbers of special type fast deterministic primality tests are known, e.g. for *Mersenne numbers* $2^n - 1$. We give no details here.

Other deterministic primality tests for arbitrary numbers have been developed, such as the Jacobi sum primality test and the elliptic curve primality test. They are usually rather time consuming, both in theory (not polynomial time) and in practice.

Until recently it was not known whether there exists a deterministic polynomial time primality proving algorithm. In 2002 three Indian computer scientists (Agrawal, Kayal and Saxena), who were unknown in the number theory community, came up with a completely new primality test, now known as the *AKS Algorithm* or *AKS primality test*, that is deterministic and runs in polynomial time. Their invention got world wide media coverage, and is a major breakthrough indeed. However, it seems that their method is too slow for practical purposes. Improvements have been found, notably by Dan Bernstein (`http://cr.yp.to`), showing quartic complexity, i.e. $\mathrm{O}((\log n)^4)$.

It is not yet clear if such algorithms will become practical for cryptographers. See [Sh, Chapter 22].

Unfortunately in this course we cannot go into details.

5.4 Prime Number Generation

5.4.1 Random Primes

The basic idea of generating primes is very simple: generate random numbers with the required properties (such as having a certain number of bits, lying in a certain interval, lying in a certain congruence class), and apply some primality test, until a (probable) prime is found. In practical cryptography often the following prime generation algorithm, based on Miller-Rabin, is used to (e.g.) generate primes of a given number of bits.

Algorithm 5.2 (Miller-Rabin Prime Number Generation)

Input:	$k \in \mathbb{N}$ (required number of bits) $R \in \mathbb{N}$ (security parameter)
Output:	a k bit probable prime p

Step 1:	repeat
	  choose a new random k bit odd integer p
	  mr $\leftarrow$ the output of the Miller-Rabin
	  primality test with input p, R
	until mr = "probably prime"

Step 2:	output p

Of practical interest is the probability that this algorithm returns a composite p. Let this probability be $p_{k,R}$. Theorem 5.9(b) suggests that $p_{k,R} < 4^{-R}$. In fact the situation is much better. We quote the following result, showing that already one run of the main loop in the Miller-Rabin primality test ($R = 1$) gives better results in practice for moderate k.

Lemma 5.12 (Damgård, Landrock and Pomerance)
If $k \geq 2$ *then* $p_{k,1} < k^2 4^{2-\sqrt{k}}$.
If $k \geq 88$ *and* $2 \leq R \leq 9$ *then* $p_{k,R} < k\sqrt{k}2^R 4^{2-\sqrt{RK}}/\sqrt{R}$.

See [MvOV, Sections 4.48, 4.49] for some more elaborate results.

In practice, when a probability of 2^{-80} is seen as acceptable and the bitsize of the primes is at least 512 (resp. 1024), a security parameter $R = 6$ (resp. 3) is already sufficient.

On present day personal computers generating primes of several thousands of bits should take at most a few seconds.

5.4.2 Strong Primes

Sometimes primes with special properties are required, such as Sophie Germain primes, or primes that are in some other sense 'strong primes'. Usually this refers to the prime divisors of $n-1$ and $n+1$, that should not all be small, to avoid certain cryptographic attacks.

Generating Sophie Germain primes is essentially slower than generating random primes. The only known way is to apply a primality test to both p and $\frac{1}{2}(p-1)$, and of course this test fails much more often than for random primes.

Algorithm 5.3 (Sophie Germain Prime Number Generation)

Input:	$k \in \mathbb{N}$ (required number of bits) $R \in \mathbb{N}$ (security parameter)
Output:	a k bit probable prime p such that $\frac{1}{2}(p-1)$ is also probable prime

Step 1:	repeat
	  choose a new random k bit odd integer p
	  mr' $\leftarrow$ the output of the Miller-Rabin
	  primality test with input $\frac{1}{2}(p-1), R$
	  if mr' = "probably prime" then
	  mr $\leftarrow$ the output of the Miller-Rabin
	  primality test with input p, R
	  else mr $\leftarrow$ "irrelevant"
	until mr = "probably prime" and mr' = "probably prime"

Step 2:	output p

The following algorithm generates primes p that are *strong primes* in the sense that $p-1$ has a prime divisor of about half the bitlength of p. For different cryptographic applications different notions of 'strong primes' may exist, so this is only an example. Variations can be easily made. The algorithm below is about as fast as random prime number generation.

Algorithm 5.4 (Strong Prime Number Generation)

Input:	$k \in \mathbb{N}$ (required number of bits) $R \in \mathbb{N}$ (security parameter)
Output:	a k bit probable prime p such that $p-1$ has a $\frac{1}{2}k$ bit probable prime divisor
Step 1:	generate a $\frac{1}{2}k$ bit probable prime q

Step 2:	repeat
	  choose a new (random) integer r
	  with $\frac{1}{q}2^{k-1} < r < \frac{1}{q}2^k$
	  $p \leftarrow qr+1$
	  mr $\leftarrow$ the output of the Miller-Rabin
	  primality test with input p, R
	until mr = "probably prime"

Step 3:	output p

5.4.3 Constructive methods

We conclude the chapter on prime numbers with mentioning that there are also constructive methods for making prime numbers. One method is based on the following result known as *Pocklington's Theorem*.

Theorem 5.13 (Pocklington's Theorem) *Let $n \geq 3$ be an odd integer. Suppose that $n = 1 + 2qR$, where q is an odd prime and $q > R$. Suppose that there exists an a such that $a^{n-1} \equiv 1 \pmod{n}$ and $\gcd(a^{2R} - 1, n) = 1$. Then n is prime.*

Proof. Assume that n is composite. Let p be an odd prime such that $p \mid n$ and $p \leq \sqrt{n}$. Let r be the order of a modulo p. Then $r \mid p-1$ by Fermat's Theorem, and $r \mid n-1$ because $a^{n-1} \equiv 1 \pmod{p}$. If $q \nmid r$ then $r \mid n-1 = 2qR$ implies $r \mid 2R$, hence $a^{2R} \equiv 1 \mod p$, hence $p | a^{2R} - 1$. But this contradicts $\gcd(a^{2R} - 1, n) = 1$. Hence $q \mid r$. This implies $q \mid p-1$, and as q is odd we must have $q \leq \frac{1}{2}(p-1)$. Finally we have $n = 1 + 2qR < 1 + 2q^2 \leq 1 + \frac{1}{2}(p-1)^2 < p^2$, and this contradicts $p \leq \sqrt{n}$. Hence n must be prime. □

The use of this theorem is as follows. Assume we have an odd prime q. Then we can try random integers $R < q$ (but not much smaller than q) and random integers a, and check the conditions of Pocklington's Theorem. If we succeed (and it can be shown that this is likely), we then have a proven prime p that is almost twice the size of q. We can then iterate this and build larger and larger primes.

Each prime generated in this way comes with a certificate, consisting of q and a, and the certificate of q. So in fact a prime comes with a certificate chain.

Note that producing a certificate may be a long process, as it is hard to predict how many choices for R and a will fail. But once a certificate is found, verifying it is easy: only two modular exponentiations are required for checking each certificate in a certificate chain.

Example: with $q = 5$ we can take $R = 3$ and $a = 7$. Then $n = 31$. Indeed, $7^{30} \equiv 1 \pmod{31}$ and $7^6 \equiv 4 \pmod{31}$. So Pocklington's Theorem shows that 31 is prime, and a certificate for the primality of 31 is $\{5, 7\}$. The primality of 5 does not need a certificate.

We can then proceed with $q = 31$, and we can take e.g. $R = 26$, $a = 1184$, and thus obtain the prime 1613 with certificate chain $\{\{5, 7\}, \{31, 1184\}\}$. Going a few steps further, we easily find that 4999499696151619140572559421 is a prime, with certificate chain
$\{\{5, 7\}, \{31, 1184\}, \{1613, 3222331\}, \{5055143, 40018519077137\}, \{50720746959643, 485133230674386845788396116\}\}$.
And we easily could have extended this a few more steps.

To conclude we note the drawbacks of this method: it cannot produce primality proofs for given numbers, it will generally be slower than probabilistic primality generating algorithms, and the primes that are produced are not really random.

Exercises

5.1. Make a list of all primes below 100. Do this as follows: write all natural numbers up to 100 in a table of e.g. 10 rows and 5 columns. Cross out 1, then cross out all multiples of 2 (except 2 itself), all multiples of 3 (except 3 itself), etcetera. Stop when you are certain that all numbers left are primes.
Describe a similar procedure to list all primes below x. The multiples of which numbers have to be crossed out? When can you stop and why? Can you estimate the complexity?
This procedure is called the *Sieve of Eratosthenes*.

5.2. Assuming the Prime Number Theorem, prove that $\pi(x) \sim \dfrac{x}{\log x - 1}$, and $\pi(x) \sim \mathrm{li}(x)$.

5.3. For each $k \in \mathbb{N}$, find a set of k consecutive composite integers.

5.4. Prove Wilson's Theorem: For $n \geq 2$, n is prime if and only if $(n-1)! \equiv -1 \pmod{n}$. Why does this not give a good primality test?

5.5. Prove Lemma 5.5.

5.6. Show that 561 is a Carmichael number. Hint: use $561 = 3 \times 11 \times 17$.

5.7. Let n be a composite number with a being a strong witness but not a Fermat witness. Then show how to find some nontrivial factors of n.

5.8. Let n be odd, and a coprime to n. Lemma 4.2(a) asserts that $\left(\dfrac{a}{n}\right) \equiv a^{(n-1)/2} \pmod{n}$ when n is prime.
Compute $\left(\dfrac{13}{561}\right)$ and $13^{280} \pmod{561}$ (you may use a computer). What is your conclusion on the primality of 561?
The above should suggest a probabilistic primality test to you. Write it out. It is known as the

Solovay-Strassen primality test.

5.9. Witnesses, liars and pseudoprimes for the Solovay-Strassen primality test (see Exercise 5.8) are called *Euler witnesses*, *Euler liars* and *Euler pseudoprimes*. Write out their definitions. Show that a strong liar is always a Fermat liar, and that an Euler liar is always a Fermat liar (in fact, a strong liar always is an Euler liar, but proving this is not asked here, see [BS, Theorem 9.3.10]). Conclude which of the tests of Miller-Rabin, Solovay-Strassen and the test based on Fermat's Theorem is the strongest. Finally prove that if $n \equiv 3 \pmod 4$ then an Euler liar always is a strong liar.

5.10. Let $n = 493$, and take $a = 2, 30, 86$ or 157. For each of these a find out whether it is a Fermat liar or witness, an Euler liar or witness, a strong liar or witness.

5.11. Prove Lemma 5.11.

Chapter 6

Multiplicative functions

Introduction

General references for this chapter: [Sh, Section 2.6]. And for those preferring Dutch: [Be, Chapters 3, 6], [Ke, Chapter 8].

In this chapter the following topics will be treated:

- arithmetic functions,
- multiplicative functions,
- the Möbius function,
- the Möbius inversion formula,
- the principle of inclusion and exclusion.

6.1 Multiplicative functions

In this section we study *arithmetic functions*, that are functions f defined on $\mathbb{N}$ that satisfy $f(1) \neq 0$. These functions may take values in any convenient subset of $\mathbb{C}$, often in $\mathbb{Z}$.

In particular we will be interested in *multiplicative functions*. An arithmetic function f is called multiplicative if it satisfies $f(mn) = f(m)f(n)$ whenever $\gcd(m, n) = 1$. Note that multiplicative functions are determined completely by their values at prime powers p^k. We already met one important multiplicative function, namely Euler's $\phi(n)$, see Theorem 3.1(a).

Recall that $\phi(n)$ satisfies $\sum_{d|n} \phi(d) = n$ (see Theorem 3.5). For many functions f on $\mathbb{N}$ the sum $g(n) = \sum_{d|n} f(d)$ is of interest. Our main goal in this section will be to invert this formula: to find an expression for $f(n)$ in terms of the above defined $g(n)$.

In order to do this in a neat way we define the *convolution product* $f * g$ of two arithmetic functions f and g, by

$$(f * g)(n) = \sum_{d|n} f\left(\frac{n}{d}\right) g(d).$$

Note that also

$$(f*g)(n) = \sum_{d|n} f(d)g\left(\frac{n}{d}\right) = \sum_{d_1 d_2 = n} f(d_1)g(d_2),$$

which is useful as it gives a more symmetric definition of convolution.

We also introduce the following easy multiplicative functions on $\mathbb{N}$:

$$E(n) = \begin{cases} 1 & \text{if} \quad n = 1 \\ 0 & \text{if} \quad n > 1 \end{cases}, \quad U(n) = 1 \text{ for all } n, \quad I(n) = n \text{ for all } n.$$

Note that we now have

$$\sum_{d|n} f(d) = (U*f)(n)$$

as a convenient notation. For example, the formula $\sum_{d|n} \phi(d) = n$ (Theorem 3.5) now amounts to $U * \phi = I$.

The following lemmas give the basic properties of the convolution product. Basically they say that the set of arithmetic functions is a commutative group with the convolution product as group operation, and that the set of multiplicative functions is a subgroup.

Lemma 6.1 *Let f, g, h be arithmetic functions.*
(a) *The convolution product $f * g$ is also an arithmetic function.*
(b) *The convolution product is commutative, i.e. $f * g = g * f$.*
(c) *The convolution product is associative, i.e. $(f * g) * h = f * (g * h)$.*
(d) *The function E is the unit for the convolution product, i.e. $f * E = E * f = f$.*
(e) *There exists a unique arithmetic function f^{-1} that is the inverse of f with respect to the convolution product, i.e. $f * f^{-1} = f^{-1} * f = E$.*

Lemma 6.2 *Let f, g be multiplicative functions.*
(a) *$f(1) = 1$.*
(b) *The convolution product $f * g$ is also a multiplicative function.*
(c) *The inverse f^{-1} is also a multiplicative function.*

Proof of Lemma 6.1.
(a) All we have to check is that $(f*g)(1) = f(1)g(1) \neq 0$.
(b) This follows by $d \mid n \Leftrightarrow \dfrac{n}{d} \mid n$.
(c) Note that

$$\begin{aligned} ((f*g)*h)(n) &= \sum_{d|n} (f*g)(d)h\left(\frac{n}{d}\right) = \sum_{dd_3=n} (f*g)(d)h(d_3) \\ &= \sum_{dd_3=n} \sum_{d_1 d_2 = d} f(d_1)g(d_2)h(d_3) = \sum_{d_1 d_2 d_3 = n} f(d_1)g(d_2)h(d_3). \end{aligned}$$

The last expression is symmetric, so it is easily seen to be equal to $(f*(g*h))(n)$.
(d) $(f*E)(n) = \sum_{d|n} f\left(\dfrac{n}{d}\right) E(d) = f(n)$.
(e) This can be done inductively (recursively) from the formula

$$(f*f^{-1})(n) = \sum_{d|n} f\left(\frac{n}{d}\right) f^{-1}(d) = E(n) = \begin{cases} 1 & \text{if} \quad n = 1 \\ 0 & \text{if} \quad n > 1 \end{cases},$$

as follows. First apply this with $n = 1$, to find $f^{-1}(1) = \dfrac{1}{f(1)}$, which is possible precisely because $f(1) \neq 0$. Next we assume that $f^{-1}(1), f^{-1}(2), \ldots, f^{-1}(n-1)$ are defined, and then from the above formula we have

$$f^{-1}(n) = -\frac{1}{f(1)} \sum_{d|n, d\neq n} f\left(\frac{n}{d}\right) f^{-1}(d).$$

As there is only one possible choice for $f^{-1}(n)$, uniqueness is guaranteed. □

Proof of Lemma 6.2.
(a) 1 is coprime to itself, so $f(1) = f(1 \cdot 1) = f(1)f(1) = f(1)^2$. As $f(1) \neq 0$ we have $f(1) = 1$.
(b) Let $m, n \in \mathbb{N}$ be coprime. The set of divisors $d \mid mn$ is in one to one correspondence to the set of pairs (d_m, d_n) such that $d_m \mid m$ and $d_n \mid n$, namely via $d = d_m d_n$. Note that d_m and d_n are coprime. So

$$\begin{aligned}
(f * g)(mn) &= \sum_{d|mn} f\left(\frac{mn}{d}\right) g(d) = \sum_{d_m|m} \sum_{d_n|n} f\left(\frac{m}{d_m}\frac{n}{d_n}\right) g(d_m d_n) \\
&= \sum_{d_m|m} \sum_{d_n|n} f\left(\frac{m}{d_m}\right) f\left(\frac{n}{d_n}\right) g(d_m) g(d_n) \\
&= \left(\sum_{d_m|m} f\left(\frac{m}{d_m}\right) g(d_m)\right) \left(\sum_{d_n|n} f\left(\frac{n}{d_n}\right) g(d_n)\right) \\
&= (f * g)(m)(f * g)(n).
\end{aligned}$$

(c) Define a multiplicative function $\hat{f}$ by $\hat{f}(p^k) = f^{-1}(p^k)$ for all primes p and $k \geq 0$ (note that this defines $\hat{f}(n)$ for all n). Then by (b) $f * \hat{f}$ is also multiplicative, and it satisfies $(f * \hat{f})(p^k) = (f * f^{-1})(p^k) = E(p^k)$. Hence $(f * \hat{f})(n) = E(n)$ for all n, i.e. $f * \hat{f} = E$, and so by the uniqueness of the inverse we have $f^{-1} = \hat{f}$. Hence f^{-1} is multiplicative. □

As examples of the recursion from the proof of Lemma 6.1(e), note that for primes p

$$\begin{aligned}
f^{-1}(p) &= -f(p), \\
f^{-1}(p^2) &= -f(p^2) + f(p)^2, \\
f^{-1}(p^3) &= -f(p^3) + 2f(p)f(p^2) - f(p)^3, \\
f^{-1}(p^4) &= -f(p^4) + 2f(p^3)f(p) + f(p^2)^2 + 2f(p^2)f(p)^2 + f(p)^4,
\end{aligned}$$

etc.

6.2 The Möbius function

The next step is to find the inverse for the function U. This function U^{-1} is commonly called μ. Applying the recursion, we find for all primes p

$$\begin{aligned}
\mu(p) &= -U(p)\mu(1) = -\mu(1) = -1, \\
\mu(p^2) &= -U(p^2) - U(p)\mu(p) = -\mu(1) - \mu(p) = 0, \\
\mu(p^3) &= -U(p^3) - U(p^2)\mu(p) - U(p)\mu(p^2) = -\mu(1) - \mu(p) - \mu(p^2) = 0, \\
\mu(p^4) &= -U(p^4) - U(p^3)\mu(p) - U(p^2)\mu(p^2) - U(p)\mu(p^3) = \\
&= -\mu(1) - \mu(p) - \mu(p^2) - \mu(p^3) = 0,
\end{aligned}$$

etc. With induction we get $\mu(p^k) = 0$ whenever $k \geq 2$, namely

$$\mu(p^k) = -\sum_{i=0}^{k-1} U(p^{k-i})\mu(p^i) = -\mu(1) - \mu(p) - \sum_{i=2}^{k-1} \mu(p^i) = -1 - (-1) - \sum_{i=2}^{k-1} 0 = 0.$$

Note that by Lemma 6.2(c) μ is multiplicative, which shows how to define $\mu(n)$ for all n.

A number $n \in \mathbb{N}$ is called *squarefree* if it is not divisible by a square other than 1. It is now clear that $\mu(n) = 0$ whenever n is not squarefree, and if n is squarefree, then $\mu(n) = 1$ or -1, according to the number of prime divisors of n being even or odd.

The function μ is called the *Möbius function*. Summarizing, it can be defined by

$$\mu(n) = \begin{cases} 0 & \text{if } n \text{ is not squarefree,} \\ 1 & \text{if } n \text{ is squarefree and has an even number of prime divisors,} \\ -1 & \text{if } n \text{ is squarefree and has an odd number of prime divisors,} \end{cases}$$

is multiplicative, and satisfies $\mu * U = E$. This last property written out reads

$$\sum_{d|n} \mu(d) = \begin{cases} 1 & \text{if} \quad n = 1 \\ 0 & \text{if} \quad n > 1 \end{cases}.$$

Example: $\mu(15) = 1$ because $15 = 3 \times 5$ is squarefree and has 2 prime divisors, $\mu(30) = -1$ because $30 = 2 \times 3 \times 5$ is squarefree and has 3 prime divisors, and $\mu(60) = 0$ because $60 = 2^2 \times 3 \times 5$ is not squarefree.

6.3 Möbius inversion

Now we reach the main result of this section, the *Möbius Inversion Formula*.

Theorem 6.3 (Möbius Inversion Formula)
Let f be an arithmetic function. Define the arithmetic function g by $g = U * f$, i.e.

$$g(n) = \sum_{d|n} f(d) \quad \textit{for all} \quad n \in \mathbb{N}.$$

Then $f = \mu * g$, that is

$$f(n) = \sum_{d|n} \mu\left(\frac{n}{d}\right) g(d) \quad \textit{for all} \quad n \in \mathbb{N}.$$

Moreover, if f is multiplicative, then so is g.

Proof. By the definition of μ we have $\mu * g = \mu * (U * f) = (\mu * U) * f = E * f = f$. Lemma 6.2 guarantees multiplicativity of g based on that of f. □

Example: with $f = E$ we have $g = U * E = U$, and Möbius inversion now gives $E = \mu * U$, which is exactly the definition of μ as the inverse of U.

Example: Theorem 3.5 showed that $\sum_{d|n} \phi(d) = n$, or, in our new language, $U * \phi = I$. When we apply Möbius inversion we get $\phi = \mu * I$, or

$$\phi(n) = \sum_{d|n} \mu(d) \frac{n}{d}.$$

This gives a new proof for Lemma 3.9. Because U and μ are multiplicative, so is ϕ. This is a new proof for Theorem 3.1(a). Now note that for primes p and $i \geq 0$ we have $\mu(p^i) = 1$ if $i = 0$, $\mu(p^i) = -1$ if $i = 1$, and $\mu(p^i) = 0$ if $i \geq 2$, so for all $k \in \mathbb{N}$

$$\phi(p^k) = \sum_{d|p^k} \mu(d) \frac{p^k}{d} = \sum_{i=0}^{k} \mu(p^i) p^{k-i} = p^k - p^{k-1}.$$

With the multiplicativity of ϕ this provides a new proof of Theorem 3.1(b).

These observations on ϕ will return in the next section.

6.4 The Principle of Inclusion and Exclusion

Consider $n = pq$ where p, q are distinct prime numbers. We will count the number of elements of $\mathbb{Z}_n^*$ by a combinatorial argument. We start with $\mathbb{Z}_n = \{0, 1, 2, \ldots, n-1\}$, which has n elements. Then we note that we should not count all multiples of p, of which there are q (namely $0, p, 2p, \ldots, (q-1)p$), and similarly we should not count all multiples of q, of which there are p (namely $0, q, 2q, \ldots, (p-1)q$). Finally we should note that we now have subtracted twice the numbers that are divisible by both p and q, of which there is exactly one (namely 0), and to compensate we should again add this number. Hence $\phi(n) = n - p - q + 1$. This is in accordance with Theorem 3.1(b).

A bit more abstract, let S be a finite set with subsets S_1 and S_2. Then the number of elements of S that are not in either one of S_1 or S_2 is equal to

$$\#(S \backslash (S_1 \cup S_2)) = \#S - \#S_1 - \#S_2 + \#(S_1 \cap S_2).$$

And with three subsets S_1, S_2, S_3 we get

$$\begin{aligned} \#(S \backslash (S_1 \cup S_2 \cup S_3)) &= \#S - \#S_1 - \#S_2 - \#S_3 + \#(S_1 \cap S_2) + \\ &\quad \#(S_1 \cap S_3) + \#(S_2 \cap S_3) - \#(S_1 \cap S_2 \cap S_3). \end{aligned}$$

In general, say that we have a finite set S with $\#S = N$. Further we have properties $P_1, P_2, \ldots, P_k$ which elements of S may or may not possess. For given $\{i_1, i_2, \ldots, i_m\} \in \{1, 2, \ldots, k\}$ with all i_j different we denote by $N_{i_1,\ldots,i_m}$ the number of elements of S that possess properties $P_{i_1}, P_{i_2}, \ldots, P_{i_m}$, but no other properties from $P_1, P_2, \ldots, P_k$. And the number of elements of S that possess none of the properties P_i is denoted by $N_\emptyset$.

Theorem 6.4 (Inclusion-Exclusion Principle)

$$\begin{aligned} N_\emptyset &= N - \sum_{1 \leq i_1 \leq k} N_{i_1} + \sum_{1 \leq i_1 < i_2 \leq k} N_{i_1,i_2} + \ldots + (-1)^k N_{1,2,\ldots,k} \\ &= N + \sum_{m=1}^{k} (-1)^m \sum_{1 \leq i_1 < \ldots < i_m \leq k} N_{i_1,\ldots,i_m}. \end{aligned}$$

Proof. Consider an element of S that satisfies none of the properties. It is counted once on both sides of the inequality.
Next consider an element of S that satisfies exactly r of the properties, with $1 \le r \le k$. On the left hand side it is not counted, and on the right hand side, in the term $\sum_{1\le i_1<\ldots<i_m\le k} N_{i_1,\ldots,i_m}$ it is counted $\binom{r}{m}$ times (which equals 0 when $r < m$). Together this amounts to $1+\sum_{m=1}^{r}(-1)^m\binom{r}{m} = (1-1)^r = 0$ times. □

Finally we will once more prove Theorem 3.1(b), and show the relation to Möbius inversion. Let n have the prime factorization $n = \prod_{i=1}^{k} p_i^{e_i}$. Let P_i be the property of divisibility by p_i, applied to elements of $\mathbb{Z}_n$. The Inclusion-Exclusion Principle then yields

$$\phi(n) = n - \sum_{1\le i_1\le k}\frac{n}{p_{i_1}} + \sum_{1\le i_1<i_2\le k}\frac{n}{p_{i_1}p_{i_2}} + \ldots + (-1)^k\frac{n}{p_1p_2\ldots p_k}.$$

On the one hand,

$$n - \sum_{1\le i_1\le k}\frac{n}{p_{i_1}} + \sum_{1\le i_1<i_2\le k}\frac{n}{p_{i_1}p_{i_2}} + \ldots + (-1)^k\frac{n}{p_1p_2\ldots p_k} = n\left(1-\frac{1}{p_1}\right)\ldots\left(1-\frac{1}{p_k}\right),$$

as can be seen by expanding the product in the right hand side. This gives exactly Theorem 3.1(b). And on the other hand every squarefree divisor of n is of the form $p_{i_1}\ldots p_{i_m}$, so

$$n - \sum_{1\le i_1\le k}\frac{n}{p_{i_1}} + \sum_{1\le i_1<i_2\le k}\frac{n}{p_{i_1}p_{i_2}} + \ldots + (-1)^k\frac{n}{p_1p_2\ldots p_k} = \sum_{d|n}\mu(d)\frac{n}{d},$$

i.e. $\phi(n) = \sum_{d|n}\mu(d)\frac{n}{d}$, in short $\phi = \mu * I$, which we had already seen in the previous section. Again we have a new proof of $\sum_{d|n}\phi(n) = n$.

6.5 Fermat and Euler revisited

The usual way to generalize Fermat's Little Theorem 3.2 to non-prime moduli is Euler's Theorem 3.1. Another generalization of Fermat's Little Theorem to non-prime moduli is the following elementary result, that seems to be not very well known.

Theorem 6.5 (Generalization of Fermat's Little Theorem) *Let $n > 1$ be an integer, and let $a \in \mathbb{Z}$. Then*

$$a^n \equiv -\sum_{k|n,k<n}\mu(n/k)a^k \pmod{n}. \tag{6.1}$$

With n having prime factors $p_1, p_2, \ldots, p_r$ (i.e. $n = \prod_{i=1}^{r} p_i^{e_i}$), we also can phrase (6.1) as

$$a^n \equiv \sum_{k=1}^{r}(-1)^{k+1}\sum_{\substack{S\subset\{1,2,\ldots,r\}\\ \#S=k}} a^{n/\prod_{i\in S}p_i} \pmod{n}.$$

To get a feeling for this, let's write out a few cases.

When $n = p$ is prime, (6.1) reads

$$a^p \equiv a \pmod{p},$$

which is just Fermat's Little Theorem.

When $n = pq$ for p, q different primes, (6.1) reads

$$a^n \equiv a^p + a^q - a \pmod{n}.$$

This statement seems already to be not well known. Note that we can infer the funny formula

$$a^{p+q-1} \equiv a^p + a^q - a \pmod{pq}.$$

When $n = pqr$ for p, q, r different primes, (6.1) reads

$$a^n \equiv a^{pq} + a^{pr} + a^{qr} - a^p - a^q - a^r + a \pmod{n}.$$

Again we get a funny formula:

$$a^{pq+pr+qr-p-q-r+1} \equiv a^{pq} + a^{pr} + a^{qr} - a^p - a^q - a^r + a \pmod{pqr}.$$

Continuation is obvious.

With $n = p^e$ the Generalization of Fermat's Little Theorem gives $a^{p^e} \equiv a^{p^{e-1}} \pmod{p^e}$, hence if $p \nmid a$ we have $a^{p^e - p^{e-1}} \equiv 1 \pmod{p^e}$, and Euler's Theorem now readily follows as well.

Here is a short proof of the above Generalization of Fermat's Little Theorem.

Proof. Consider sequences $(x_1, x_2, \ldots, x_n)$ of elements $x_i \in \mathbb{Z}_a$. Clearly there are a^n possible sequences. We define two sequences to be equivalent when the one is a cyclic shift of the other. Let α_k be the number of equivalence classes consisting of k elements. Clearly α_k is nonzero precisely when $k|n$, and we find

$$a^n = \sum_{k|n} k\alpha_k.$$

Möbius inversion yields

$$n\alpha_n = \sum_{k|n} \mu(n/k) a^k,$$

and this immediately gives the result. □

Exercises

6.1. Let $f : \mathbb{N} \to \mathbb{Z}$ be an arithmetic function. Show that if f is multiplicative, then also f^{-1} is a function from $\mathbb{N}$ to $\mathbb{Z}$. What if f is not multiplicative?

6.2. Give interpretations for $U * U$ and $U * I$.

6.3. Compute the inverse I^{-1} of I. Also show that the inverse ϕ^{-1} of ϕ is given by $\sum_{d|n} d\mu(d)$.

6.4. Count the number of positive integers below 1000 that are not divisible by $5, 6$ or 7.

Chapter 7

Continued Fractions

Introduction

General reference for this chapter: [HW, Chapters XI and XXIII]. And for those preferring Dutch: [Be, Chapter 14], [Ke, Sections 7.6, 15.6, 15.7].

In this chapter the following topics will be treated

- continued fractions,
- lattice basis reduction,
- diophantine approximation.

7.1 The Euclidean Algorithm revisited

We reformulate the Extended Euclidean Algorithm. Say we want to apply it to the positive integers s and t. Then we put

$$s_0 = t_{-1} = s, \quad t_0 = t, \quad p_{-2} = 0, \quad p_{-1} = 1, \quad q_{-2} = 1, \quad q_{-1} = 0,$$

and then do the following computation for $n = 0, 1, 2, \ldots$ until we reach $t_n = 0$:

$a_n = \left\lfloor \frac{s_n}{t_n} \right\rfloor$,
$p_n = a_n p_{n-1} + p_{n-2}$,
$q_n = a_n q_{n-1} + q_{n-2}$,
$s_{n+1} = t_n$,
$t_{n+1} = s_n - a_n t_n$.

The invariant now is $p_n t - q_n s = (-1)^{n+1} t_{n+1} = (-1)^{n+1} s_{n+2}$. We will not be interested anymore in the greatest common divisor, but in the numbers a_n, p_n and q_n that show up in this algorithm. Note that they depend on the fraction, and are independent of the gcd: if s and t are multiplied by the same integer, then the a_n, p_n, q_n do not change at all.

Example: let us take $s = 23$ and $t = 16$. Then

n	-2	-1	0	1	2	3	4
s_n			23	16	7	2	1
t_n		23	16	7	2	1	0
a_n			1	2	3	2	
p_n	0	1	1	3	10	23	
q_n	1	0	1	2	7	16	

7.2 Continued Fractions

If we write the successive divisions with remainder really as divisions, we get

$$\frac{23}{16} = 1 + \frac{7}{16}, \quad \frac{16}{7} = 2 + \frac{2}{7}, \quad \frac{7}{2} = 1 + \frac{1}{2}, \quad \frac{2}{1} = 2.$$

Note that the fractions are chained, each time put upside down. When we substitute each next fraction in the previous one, we get

$$\frac{23}{16} = 1 + \frac{7}{16} = 1 + \frac{1}{\frac{16}{7}} = 1 + \frac{1}{2 + \frac{2}{7}} = 1 + \frac{1}{2 + \frac{1}{\frac{7}{2}}} = 1 + \frac{1}{2 + \frac{1}{3 + \frac{1}{2}}}.$$

In general we get

$$\frac{s}{t} = \frac{s_0}{t_0} = a_0 + \frac{t_1}{s_1} = a_0 + \frac{1}{\frac{s_1}{t_1}} = a_0 + \frac{1}{a_1 + \frac{1}{\frac{s_2}{t_2}}} = \ldots = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{\ddots + \frac{1}{a_n}}}}}.$$

This is a typographical nightmare, that's why we introduce the notation

$$[a_0, a_1, a_2, \ldots, a_n] = a_0 + \cfrac{1}{a_1 + \cfrac{1}{\ddots + \frac{1}{a_n}}}.$$

Example: $\dfrac{23}{16} = [1, 2, 3, 2]$.

Next we look at the truncations:

$$[1] = 1 = \frac{1}{1}, \quad [1, 2] = 1 + \frac{1}{2} = \frac{3}{2}, \quad [1, 2, 3] = 1 + \frac{1}{2 + \frac{1}{3}} = \frac{10}{7}, \quad [1, 2, 3, 2] = \frac{23}{16}.$$

It looks like

$$[a_0, a_1, a_2, \ldots, a_n] = \frac{p_n}{q_n}$$

for all n, and this is indeed true in general, as we'll see below.

Expressions like $[a_0, a_1, a_2, \ldots]$ are called *continued fractions*. We now introduce the continued fraction algorithm. When applied for rational numbers, it is just a restatement of the Extended Euclidean Algorithm applied to numerator and denominator. But it can also be applied for non-rational numbers. The algorithm is the same: each time take the integral part, put the remainder upside down and continue with that.

Take for example $\pi = 3.14159\ldots$, then we get

$$\pi = 3 + 0.14159\ldots, \quad \tfrac{1}{0.14159\ldots} = 7 + 0.06251\ldots, \quad \tfrac{1}{0.06251\ldots} = 15 + 0.99659\ldots,$$
$$\tfrac{1}{0.99659\ldots} = 1 + 0.00341\ldots, \quad \tfrac{1}{0.00341\ldots} = 292 + 0.63459\ldots, \quad \text{etc.}$$

So $\pi = [3, 7, 15, 1, 292, \ldots]$.

Rational numbers have finite continued fractions (why?), irrational numbers have infinite continued fractions (why?).

When $\alpha = [a_0, a_1, a_2, \ldots]$ (finite or infinite), the fractions that are equal to the truncated continued fractions, i.e. $\dfrac{p_n}{q_n} = [a_0, a_1, \ldots, a_n]$, are called *convergents* of α. The numbers $a_0, a_1, a_2, \ldots$ are called *partial quotients* of the continued fraction.

The following algorithm computes partial quotients and convergents for any $\alpha \in \mathbb{R}_{>0}$ (for negative numbers everything goes through as well).

Algorithm 7.1 (Continued Fraction Algorithm)

Input:	$\alpha \in \mathbb{R}_{>0}$, $n \in \mathbb{N}$
Output:	`truncated continued fraction` $[a_0, a_1, a_2, \ldots, a_m]$ `of` α `and the convergents` $\frac{p_0}{q_0}, \frac{p_1}{q_1}, \ldots, \frac{p_m}{q_m}$, `where` $m = n$ `unless` $\alpha \in \mathbb{Q}$ `and` $\alpha = [a_0, a_1, \ldots, a_m]$ `with` $m < n$
Step 1:	$\alpha_0 \leftarrow \alpha$ $p_{-2} \leftarrow 0$, $p_{-1} \leftarrow 1$ $q_{-2} \leftarrow 1$, $q_{-1} \leftarrow 0$ $m \leftarrow n$
Step 2:	`for` i `from` 0 `to` m `do` $a_i \leftarrow \lfloor \alpha_i \rfloor$ $p_i \leftarrow a_i p_{i-1} + p_{i-2}$ $q_i \leftarrow a_i q_{i-1} + q_{i-2}$ `if` $\alpha_i = a_i$ `then` $m \leftarrow i$ `if` $i < m$ `then` $\alpha_{i+1} \leftarrow \dfrac{1}{\alpha_i - a_i}$
Step 3:	`output` $[a_0, a_1, \ldots, a_m], \dfrac{p_0}{q_0}, \dfrac{p_1}{q_1}, \ldots, \dfrac{p_m}{q_m}$

Note that usually $m = n$, except in the case where $\alpha \in \mathbb{Q}$ has a continued fraction that is shorter than n partial quotients.

Let us show that the convergents can indeed be found by the recurrence formulas as given in the continued fraction algorithm.

Lemma 7.1 *Let $\alpha = [a_0, a_1, a_2, \ldots]$. Let p_n, q_n for $n = -2, -1, 0, 1, 2, \ldots$ be given by*

$$\begin{cases} p_{-2} = 0, & p_{-1} = 1, & p_n = a_n p_{n-1} + p_{n-2} \\ q_{-2} = 1, & q_{-1} = 0, & q_n = a_n q_{n-1} + q_{n-2} \end{cases} \quad \text{for } n = 0, 1, 2, \ldots.$$

Then $\dfrac{p_n}{q_n} = [a_0, a_1, \ldots, a_n]$, i.e. the $\dfrac{p_n}{q_n}$ are the convergents of α.

To prove this result we need the following auxiliary lemma.

Lemma 7.2 *Let $a_0 \in \mathbb{Z}$, and $a_1, a_2, \ldots \in \mathbb{N}$ (finitely or infinitely many). Let p_n, q_n for $n = 0, 1, 2, \ldots$ be given as in Lemma 7.1. Then for all $\xi \in \mathbb{R}$, $\xi > 0$ and all $n \geq 0$*

$$\frac{\xi p_n + p_{n-1}}{\xi q_n + q_{n-1}} = [a_0, a_1, \ldots, a_n, \xi].$$

Further $p_nq_{n-1} - p_{n-1}q_n = (-1)^{n-1}$ *for all* $n \geq -1$, *and* $p_nq_{n-2} - p_{n-2}q_n = (-1)^n a_n$ *for all* $n \geq 0$.

Proof. By induction. For $n = 0$ we have $\dfrac{\xi p_0 + p_{-1}}{\xi q_0 + q_{-1}} = \dfrac{\xi a_0 + 1}{\xi} = a_0 + \dfrac{1}{\xi} = [a_0, \xi]$. Next assume that $\dfrac{\eta p_{n-1} + p_{n-2}}{\eta q_{n-1} + q_{n-2}} = [a_0, a_1, \ldots, a_{n-1}, \eta]$ for some $n \geq 1$ and all $\eta \in \mathbb{R}$, $\eta > 0$. Then we find

$$\begin{aligned}[a_0, a_1, \ldots, a_n, \xi] &= \left[a_0, a_1, \ldots, a_{n-1}, a_n + \frac{1}{\xi}\right] = \frac{\left(a_n + \frac{1}{\xi}\right) p_{n-1} + p_{n-2}}{\left(a_n + \frac{1}{\xi}\right) q_{n-1} + q_{n-2}} \\ &= \frac{a_n p_{n-1} + p_{n-2} + \frac{1}{\xi} p_{n-1}}{a_n q_{n-1} + q_{n-2} + \frac{1}{\xi} q_{n-1}} = \frac{p_n + \frac{1}{\xi} p_{n-1}}{q_n + \frac{1}{\xi} q_{n-1}} = \frac{\xi p_n + p_{n-1}}{\xi q_n + q_{n-1}}.\end{aligned}$$

The proofs of $p_nq_{n-1} - p_{n-1}q_n = (-1)^{n-1}$ and $p_nq_{n-2} - p_{n-2}q_n = (-1)^n a_n$ are left as an exercise (7.6). □

Proof of Lemma 7.1. Apply Lemma 7.2 with $\xi = a_n$, to find

$$[a_0, a_1, \ldots, a_{n-1}, a_n] = \frac{a_n p_{n-1} + p_{n-2}}{a_n q_{n-1} + q_{n-2}} = \frac{p_n}{q_n}.$$

□

A consequence is that the denominators q_n of the convergents grow at least exponentially, see Exercise 7.7. This implies that finding the convergents up to a certain large size of the numerator and denominator is computationally easy.

We return to the example of $\pi = [3, 7, 15, 1, 292, \ldots]$, and give the first few convergents:

$\dfrac{p_0}{q_0} = 3$, $\dfrac{p_1}{q_1} = \dfrac{22}{7} = 3.14285\ldots$, $\dfrac{p_2}{q_2} = \dfrac{333}{106} = 3.14150\ldots$, $\dfrac{p_3}{q_3} = \dfrac{355}{113} = 3.14159\ldots$.

This might clarify the term convergents. In Greek Antiquity the approximations $\dfrac{22}{7}$ and $\dfrac{355}{113}$ were already known as good approximations to π.

7.3 Diophantine approximation

Diophantine approximation is the area of number theory that studies how well real numbers can be approximated by rational numbers. Continued fractions play an important role here.

Theorem 7.3 (Inequality for convergents) *Let* $\dfrac{p_n}{q_n}$ *be a convergent of* $\alpha = [a_0, a_1, a_2, \ldots]$. *Then*

$$\frac{1}{(a_{n+1} + 2)q_n^2} < \left|\alpha - \frac{p_n}{q_n}\right| < \frac{1}{a_{n+1} q_n^2}.$$

Proof. Define $\alpha_{n+1} = [a_{n+1}, a_{n+2}, \ldots]$ so that $\alpha = [a_0, a_1, a_2, \ldots, a_n, \alpha_{n+1}]$. Lemma 7.2 with $\xi = \alpha_{n+1}$ gives

$$\left|\alpha - \frac{p_n}{q_n}\right| = \left|\frac{\alpha_{n+1} p_n + p_{n-1}}{\alpha_{n+1} q_n + q_{n-1}} - \frac{p_n}{q_n}\right| = \frac{|p_{n-1}q_n - p_n q_{n-1}|}{(\alpha_{n+1} q_n + q_{n-1}) q_n} = \frac{1}{(\alpha_{n+1} q_n + q_{n-1}) q_n},$$

and the result now follows by $a_{n+1} = \lfloor \alpha_{n+1} \rfloor$, because $a_{n+1}q_n < a_{n+1}q_n + q_{n-1} < \alpha_{n+1}q_n + q_{n-1} < (a_{n+1}+1)q_n + q_{n-1} < (a_{n+1}+2)q_n$. □

The following corollaries are immediate.

Theorem 7.4 (Convergents converge) *Let* $\frac{p_0}{q_0}, \frac{p_1}{q_1}, \frac{p_2}{q_2}, \ldots$ *be the convergents of* $\alpha \notin \mathbb{Q}$. *Then* $\alpha = \lim_{n\to\infty} \frac{p_n}{q_n}$.

Theorem 7.5 (Necessary condition for convergents) *Let* $\frac{p}{q}$ *be a convergent of* α. *Then* $\left|\alpha - \frac{p}{q}\right| < \frac{1}{q^2}$.

Theorem 7.6 (Diophantine Approximation) *If* $\alpha \notin \mathbb{Q}$ *the inequality* $\left|\alpha - \frac{p}{q}\right| < \frac{1}{q^2}$ *has infinitely many solutions* $\frac{p}{q}$.

Theorem 7.3 shows that the convergents approximate the number α very well. When a large partial quotient occurs, the previous convergent is an extremely good approximation. This becomes clear in the example of π, where the approximations $\frac{22}{7}$ and $\frac{355}{113}$, already well known in antiquity[1], indeed correspond to large partial quotients, namely 15 and 292.

The following results give a criterium for an approximation for being a convergent (a converse to Theorem 7.5), and show that convergents are exactly the best approximations in some sense. First some definitions.

A rational number $\frac{p}{q}$ is called a *best approximation* to α if all rational numbers which are closer to α have larger numerator and denominator. In other words, $\frac{p}{q}$ is a best approximation to α if for all $\frac{p'}{q'}$ with $\left|\alpha - \frac{p'}{q'}\right| < \left|\alpha - \frac{p}{q}\right|$ it holds that $q' > q$.

A rational number $\frac{p}{q}$ is called a *strong best approximation* to α if for all $\frac{p'}{q'}$ with $|q'\alpha - p'| < |q\alpha - p|$ it holds that $q' > q$.

A strong best approximation always is a best approximation, but not necessarily the other way around, see Exercise 7.15. The concept of best approximation seems better from an intuitive point of view, but the concept of strong best approximation appears to have nicer properties.

Theorem 7.7 (Sufficient condition for convergents) *Let* $\alpha \in \mathbb{R}$, *and let* $\frac{p}{q}$ *be a rational number satisfying* $\left|\alpha - \frac{p}{q}\right| < \frac{1}{2q^2}$. *Then* $\frac{p}{q}$ *is a convergent of* α.

Theorem 7.8 (Convergents are Strong Best Approximations) *Let* $\alpha \in \mathbb{R}$. *The rational number* $\frac{p}{q}$ *is a convergent of* α *if and only if it is a strong best approximation to* α.

To prove these results we introduce a Lemma.

[1] $\frac{22}{7}$ as good approximation to π was known to Archimedes (Greece, 3rd century BC), and $\frac{355}{113}$ to Zu Chongzhi (China, 5th century AD, the first European appearance is from Adriaan Anthoniszoon, The Netherlands, 1585).

Lemma 7.9 *Let $\alpha = [a_0, a_1, a_2, \ldots]$ have convergents $\frac{p_i}{q_i}$ for $i = 0, 1, 2, \ldots$. Let $\frac{p}{q}$ be a rational number such that $q_{n-1} < q < q_n$ for some $n \in \mathbb{N}$. Then $|p - q\alpha| > |p_{n-1} - q_{n-1}\alpha|$.*

Proof. Let x, y be the solution of the system

$$\begin{cases} p_{n-1}x + p_n y &= p \\ q_{n-1}x + q_n y &= q \end{cases},$$

in other words, using $p_n q_{n-1} - p_{n-1}q_n = (-1)^{n-1}$, we take

$$x = (-1)^n(pq_n - p_n q), \quad y = (-1)^{n-1}(pq_{n-1} - p_{n-1}q),$$

and we see that both are integers. When $y = 0$ we find that $p = xp_{n-1}, q = xq_{n-1}$ with $x > 1$, so $|p - q\alpha| = x|p_{n-1} - q_{n-1}\alpha| > |p_{n-1} - q_{n-1}\alpha|$. If $y > 0$, then $xq_{n-1} = q - yq_n < q - q_n < 0$, hence $x < 0$. And if $y < 0$, then $xq_{n-1} = q - yq_n > q$, hence $x > 0$. So x and y have opposite sign. Next we note that $p_{n-1} - q_{n-1}\alpha$ and $p_n - q_n\alpha$ also have opposite sign, see Exercise 7.8. So $(p_{n-1} - q_{n-1}\alpha)x$ and $(p_n - q_n\alpha)y$ have equal sign. Now we get

$$\begin{aligned} |p - q\alpha| &= |(p_{n-1} - q_{n-1}\alpha)x + (p_n - q_n\alpha)y| = |(p_{n-1} - q_{n-1}\alpha)x| + |(p_n - q_n\alpha)y| \\ &> |p_{n-1} - q_{n-1}\alpha|, \end{aligned}$$

unless $x = 0$, but that would imply $q = q_n y \geq q_n$, which is not true. □

Proof of Theorem 7.8. Let $\frac{p}{q}$ be a strong best approximation of α, and let α have convergents $\frac{p_i}{q_i}$ for $i = 0, 1, 2, \ldots$. There is an index n such that $q_{n-1} \leq q < q_n$. If $q > q_{n-1}$ then lemma 7.9 shows that $|p_{n-1} - q_{n-1}\alpha| < |p - q\alpha|$, contradicting that $\frac{p}{q}$ is a strong best approximation of α. So $q = q_{n-1}$, and then $p = p_{n-1}$, so $\frac{p}{q}$ is indeed a convergent.
Now, let $\frac{p_m}{q_m}$ be a convergent of α, then we have to show that it is a strong best approximation. So let $\frac{p}{q}$ be a rational number with $q \leq q_m$, then we must show that $|p - q\alpha| \geq |p_m - q_m\alpha|$. When $q = q_m$ this is trivial. When $q < q_m$ there is an $n \leq m$ such that $q_{n-1} \leq q < q_n$. Lemma 7.9 shows that either $q = q_{n-1}$ or $|p - q\alpha| > |p_{n-1} - q_{n-1}\alpha|$, so in both cases $|p - q\alpha| \geq |p_{n-1} - q_{n-1}\alpha|$. Exercise 7.9 then shows that $|p - q\alpha| \geq |p_{n-1} - q_{n-1}\alpha| > |p_m - q_m\alpha|$. □

Proof of Theorem 7.7. Let $\frac{p'}{q'}$ be an approximation to α such that $|q'\alpha - p'| < |q\alpha - p|$. Then we have

$$\begin{aligned} 1 &\leq |qp' - pq'| \leq |qp' - qq'\alpha| + |qq'\alpha - pq'| = q|p' - q'\alpha| + q'|q\alpha - p| \\ &< (q + q')|q\alpha - p| = (q + q')q\left|\alpha - \frac{p}{q}\right| < (q + q')q\frac{1}{2q^2} = \frac{q + q'}{2q}, \end{aligned}$$

hence $q' > q$. This shows that $\frac{p}{q}$ is a strong best approximation, hence by Theorem 7.8 a convergent. □

We finally note that to compute all convergents of α with denominators up to N, the number α should be available with a precision of size at least N^{-2}. When large partial quotients occur then accordingly larger precision may be needed. See Exercise 7.16.

Exercises

7.1. Show that a finite continued fraction represents a rational number. Conversely, show that any rational number has a finite continued fraction. Show that this finite continued fraction is unique apart from the possibility $[a_0, a_1, \ldots, a_n] = [a_0, a_1, \ldots, a_n - 1, 1]$.

7.2. With p_n, q_n, s_n, t_n, s, t as in Section 7.2, show that $p_n t - q_n s = (-1)^{n+1} t_{n+1}$.

7.3. Compute a few partial quotients and convergents of $e = 2.71828\ldots$. Do you notice any pattern? If so, do not attempt to prove it, as the proof is far from trivial.

7.4. Compute the continued fraction and convergents of $\dfrac{144}{89}$. Note that 89 and 144 are consecutive Fibonacci numbers. What is the continued fraction of $\dfrac{F_{n+1}}{F_n}$ for any n?

7.5. Compute a few partial quotients and convergents of $\alpha = \frac{1}{2}(1 + \sqrt{5})$. Find the pattern and prove it. Same questions for $\alpha = \sqrt{3}$.

In Exercises 7.6 – 7.9, let $\dfrac{p_n}{q_n}$ *for* $n = 0, 1, 2, \ldots$ *be the convergents of* $\alpha \in \mathbb{R}$.

7.6. Prove that $p_n q_{n-1} - p_{n-1} q_n = (-1)^{n-1}$ for all $n \geq -1$. Hint: use induction. Next prove that $\gcd(p_n, q_n) = 1$, and that $p_n q_{n-2} - p_{n-2} q_n = (-1)^n a_n$ for all $n \geq 0$.

7.7. Prove that $q_n \geq F_n$, where F_n is the nth Fibonacci number.

7.8. Show that $\dfrac{p_0}{q_0} < \dfrac{p_2}{q_2} < \dfrac{p_4}{q_4} < \ldots < \alpha < \ldots < \dfrac{p_5}{q_5} < \dfrac{p_3}{q_3} < \dfrac{p_1}{q_1}$. Hint: use Exercise 7.6.

7.9. Prove that $\dfrac{1}{q_{n+2}} < |p_n - q_n \alpha| \leq \dfrac{1}{q_{n+1}}$. Hint: make a slight refinement in the proof of Theorem 7.3. Next show that $|p_n - q_n \alpha| > |p_{n-1} - q_{n-1} \alpha|$.

7.10. Is Theorem 7.6 true for $\alpha \in \mathbb{Q}$?

7.11. In your favourite computer language (C, Java, ...) or computer algebra system (Mathematica, Maple, ...) program the continued fraction algorithm (do not use built-in functions that do all the work for you). Compute all solutions $\dfrac{p}{q}$ to $\left|\pi - \dfrac{p}{q}\right| < \dfrac{1}{2q^2}$ with $q < 10^{12}$. Also compute all solutions $\dfrac{p}{q}$ to $\left|\pi - \dfrac{p}{q}\right| < \dfrac{1}{q^2}$ with $q < 10^4$.

7.12. Find all solutions $x, y \in \mathbb{Z}$ with $0 < y < 10^{12}$ satisfying $|2^x - 3^y| < \dfrac{100 \cdot 3^y}{y^2}$.

7.13. Using a computer program, compute up to some point the continued fractions of $\sqrt{d}$ for the squarefree d up to 100, and find the pattern.

7.14. Prove that if the continued fraction of α is ultimately periodic, then α is the root of a quadratic polynomial with integer coefficients.

[[The converse is also true, i.e. every such so-called *quadratic number* has an ultimately periodic continued fraction. This is known as Lagrange's theorem, it is more complicated to prove, and we do not dare to ask that from you. All kinds of patterns can be found in these periods.]]

7.15. Prove that a strong best approximation to a number α is a best approximation to α. Give an example of a best approximation to some number α that is not a strong best approximation.

7.16. Let β and γ share the first n partial quotients of their continued fraction expansions. Show that the continued fraction expansion of any number between β and γ also shares at least those first n partial quotients. Show that $|\beta - \gamma| < \dfrac{2}{q_n^2}$.

Bibliography

[Be] Frits Beukers, *Getaltheorie voor beginners*, Epsilon Uitgaven No. 42, Utrecht, 1999, 4e druk: 2008.

[BS] Eric Bach and Jeffrey Shallit, *Algorithmic Number Theory Vol. 1, Efficient Algorithms*, MIT Press, Cambridge Mass., 1996.

[CP] Richard Crandall and Carl Pomerance, *Prime Numbers, A Computational Perspective*, 2nd ed., Springer Verlag, Berlin, 2005.

[Ga] Steven Galbraith, *Mathematics of Public Key Cryptography*, version 0.9, February 11, 2011, `http://www.math.auckland.ac.nz/~sgal018/crypto-book/crypto-book.html`.

[GG] Joachim von zur Gathen and Jürgen Gerhard, *Modern Computer Algebra*, 2nd ed., Cambridge University Press, Cambridge, 2003.

[HW] G.H. Hardy and E.M. Wright, *An Introduction to the Theory of Numbers*, 6th ed., Oxford University Press, Oxford, 2008.

[Ke] Frans Keune, *Getallen - van natuurlijk naar imaginair*, Epsilon Uitgaven No. 65, Utrecht, 2009.

[Kn] Donald E. Knuth, *The Art of Computer Programming Vol. 2, Seminumerical Algorithms*, Addison-Wesley, Reading Mass., 3rd ed., 1997.

[MvOV] Alfred J. Menezes, Paul van Oorschot and Scott Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1996. An online version is available at `http://www.cacr.math.uwaterloo.ca/hac/`.

[NV] Phong Nguyen and Brigitte Vallé (eds.), *The LLL Algorithm - Survey and Applications*, Springer, 2010.

[Sh] Victor Shoup, *A Computational Introduction to Number Theory and Algebra*, 2nd ed., Cambridge University Press, Cambridge, 2008. An online version is available at `http://www.shoup.net/ntb/`.

[St] William Stein, *Elementary Number Theory: Primes, Congruences, and Secrets*, Springer, 2008, online version available at `http://modular.math.washington.edu/ent/`.

[Wa] Samuel S. Wagstaff, *Cryptanalysis of Number Theoretic Ciphers*, Chapman and Hall / CRC, 2002.

[dW] Benne de Weger, *Elementaire getaltheorie en asymmetrische cryptografie*, Epsilon Uitgaven No. 63, Utrecht, 2009. Second edition 2011. For the accompanying software, see `http://www.win.tue.nl/~bdeweger/MCR/`