Computer algebra

—

a beginners course using SAGE

Universität Hannover, WS 2012

Jürgen Müller

Contents

I Theoretical background

1 Integer arithmetic

(1.1) Asymptotic behaviour. Let $f\colon \mathbb{N} \to \mathbb{R}$ be an **eventually positive function**, that is we have $f(n) > 0$ for all $n \gg 0$. Then let the **Landau symbols** $O(f)$ and $o(f)$ be the sets of all eventually positive functions $g\colon \mathbb{N} \to \mathbb{R}$ such that the sequence $[\frac{g(n)}{f(n)}; n \gg 0] \subseteq \mathbb{R}$ is bounded, respectively such that $\lim_{n\to\infty} \frac{g(n)}{f(n)} = 0$; in particular we have $o(f) \subseteq O(f)$.

Moreover, g and f are called **asymptotically equivalent** if $\lim_{n\to\infty} \frac{g(n)}{f(n)} = 1$; we write $g \sim f$, and in particular we have $f \in O(g)$ and $g \in O(f)$. We similarly use this for functions in several variables, or for functions defined on subsets of $\mathbb{R}$ unbounded to the right.

(1.2) Representation of integers. The number of digits with respect to the base $1 \neq z \in \mathbb{N}$ necessary to represent $n = \sum_{i=0}^{b-1} n_i z^i \in \mathbb{N}$, where $n_i \in \{0, \ldots, z-1\}$, is given as the **bit length** $b = b_z(n) := 1 + \lfloor \log_z(n) \rfloor = 1 + \lfloor \frac{\ln(n)}{\ln(z)} \rfloor$, where $\lfloor \cdot \rfloor$ denotes lower Gaussian brackets. For $n \in \mathbb{Z}$ we only need an additional sign, hence the **input length** of $n \in \mathbb{Z}$ (into a Turing machine) is $1 + b_z(|n|) \sim \ln(|n|)$, where the machine representation of the number $n \in \mathbb{N}$ is just the sequence $[n_0, \ldots, n_{b-1}]$.

The computational complexity of integer arithmetic is counted in **bit operations**, that is 'and', 'or', 'exclusive or', 'not' and 'shift' on bit strings, that is with respect to the base $z = 2$. Generalized bit operations are **Byte operations**, **word operations** and **long word operations** with respect to the bases $z = 2^8$, $z = 2^{32}$ and $z = 2^{64}$, respectively. The time needed for each of these operations (on a Turing machine) is polynomial in the input length.

Hence we treat (generalized) bit operations as **oracles**. An algorithm using integer arithmetic is called a **polynomial-time** or **exponential-time**, if upon input of $n \in \mathbb{Z}$ it runs in time at most $O(\ln^c(|n|))$, respectively $O(|n|^c) = O(\exp(c \ln(|n|)))$, for some $c > 0$.

(1.3) Addition Let $n \geq m \in \mathbb{N}$ and $b := b_z(n)$, for some $1 \neq z \in \mathbb{N}$; note that $b_z(n) = \max\{b_z(n), b_z(m)\}$. Hence we have $n = \sum_{i=0}^{b-1} n_i z^i$, and we may assume $m = \sum_{j=0}^{b-1} m_j z^j$, by letting $m_j := 0$ for $j \in \{b_z(m), \ldots, b-1\}$.

- $\delta \leftarrow 0$
- for $k \in [0, \ldots, b-1]$ do
  - $s_k \leftarrow n_k + m_k + \delta$
  - if $s_k \geq z$ then
    - $s_k \leftarrow s_k - z$
    - $\delta \leftarrow 1$
  - else $\delta \leftarrow 0$

- $s_b \leftarrow \delta$
- return $[s_0, \ldots, s_b]$

Hence we have $n + m = \sum_{k=0}^{b} s_k z^k$. For each $k \in \{0, \ldots, b\}$ this needs a fixed number of bit operations, and hence needs $O(b) = O(\ln(n))$ bit operations, that is runs in **linear time**. A similar analysis shows that subtraction also needs $O(b)$ bit operations.

(1.4) Multiplication Let $n = \sum_{i=0}^{b_n-1} n_i z^i \in \mathbb{N}$ and $m = \sum_{j=0}^{b_m-1} m_j z^j \in \mathbb{N}$, where $b_n := b_z(n)$ and $b_m := b_z(m)$, for some $1 \neq z \in \mathbb{N}$. Hence we have the following formula

$$nm = \sum_{i=0}^{b_n-1} \sum_{j=0}^{b_m-1} n_i m_j z^{i+j} = \sum_{k=0}^{b_n+b_m-1} \left(\sum_{l=\max\{0,k-b_m+1\}}^{\min\{b_n-1,k\}} n_l m_{k-l} \right) \cdot z^k.$$

- for $k \in [0, \ldots, b_n + b_m - 1]$ do $s_k \leftarrow 0$
- for $i \in [0, \ldots, b_n - 1]$ do
  - $\delta \leftarrow 0$
  - for $j \in [0, \ldots, b_m - 1]$ do
    - $s \leftarrow s_{i+j} + n_i m_j + \delta$ $\quad \# \; s = (s \mod z) + \lfloor \frac{s}{z} \rfloor \cdot z$
    - $s_{i+j} \leftarrow s \mod z$
    - $\delta \leftarrow \lfloor \frac{s}{z} \rfloor$
  - $s_{i+b_m} \leftarrow \delta$
- return $[s_0, \ldots, s_{b_n+b_m-1}]$

Hence we have $nm = \sum_{k=0}^{b_n+b_m-1} s_k z^k$. For each $i \in \{0, \ldots, b_n - 1\}$ and $j \in \{0, \ldots, b_m - 1\}$ this needs a fixed number of bit operations, thus in total needs $O(b_n b_m) = O(\ln(n)\ln(m))$ bit operations; note that if $n \geq m$ then this amounts to $O(\ln^2(n))$ bit operations, that is runs in **quadratic time**.

(1.5) Fast multiplication [Karatsuba, 1962]. Let $k \in \mathbb{N}_0$ and $b := 2^k$, as well as $1 \neq z \in \mathbb{N}$ and $m, n \in \mathbb{N}$ such that $m, n < z^b$; hence we have $b_z(m), b_z(n) \leq b$.

Let $m = m' \cdot z^{\frac{b}{2}} + m''$ and $n = n' \cdot z^{\frac{b}{2}} + n''$, for some $0 \leq m', m'', n', n'' < z^{\frac{b}{2}}$. Hence we have $0 \leq |m' - m''|, |n' - n''| < z^{\frac{b}{2}}$ as well, and moreover

$$m \cdot n = m'n' z^b + (m'n'' + m''n') \cdot z^{\frac{b}{2}} + m''n'',$$

where $m'n'' + m''n' = m'n' + m''n'' + (m' - m'')(n'' - n')$. Thus we let $K(m, n, k)$ be the following algorithm:

- if $k = 0$ then return mn
- else
  - $r \leftarrow K(m', n', k - 1)$
  - $s \leftarrow K(m'', n'', k - 1)$
  - $t \leftarrow K(|m' - m''|, |n' - n''|, k - 1)$

- $b \leftarrow 2^k$
- return $rz^b + (r + s \pm t) \cdot z^{\frac{b}{2}} + s$

Hence by induction with respect to $k \in \mathbb{N}_0$ we have $K(m, n, k) = mn$. We show that this **divide and conquer** technique needs $O(b^{\log_2(3)})$ bit operations; assuming that $n \geq m$ and $\frac{b}{2} < b_z(n) \leq b$ this amounts to $O(\ln^{\log_2(3)}(n))$ bit operations, where $1 < \frac{158}{100} < \log_2(3) < \frac{159}{100} < 2$, thus the Karatsuba Algorithm runs in time strictly between quadratic and linear:

Let $\kappa(k) \in \mathbb{N}$ be the number of bit operations needed to compute $K(\cdot, \cdot, k)$. Then we may assume that $\kappa(0) = 1$, and for $k > 0$ we have three calls of $K(\cdot, \cdot, k-1)$ as well as additions and shifts, thus $\kappa(k) = 3 \cdot \kappa(k-1) + \gamma \cdot b$, for some $\gamma > 0$, where $b = 2^k$. Thus by induction we get

$$\kappa(k) = 3^k \cdot \kappa(0) + \gamma \cdot \sum_{i=0}^{k-1} (3^i \cdot 2^{k-i}) = 3^k + 2^k \cdot \gamma \cdot \frac{(\frac{3}{2})^k - 1}{\frac{3}{2} - 1} = (2\gamma + 1) \cdot 3^k - \gamma \cdot 2^{k+1}.$$

Hence we have $\kappa(k) \in O(3^k) = O(3^{\log_2(b)}) = O((2^{\log_2(3)})^{\log_2(b)}) = O(b^{\log_2(3)})$. ♯

The best known algorithm for integer multiplication, the **Schönhage-Strassen Algorithm [1971]**, based on **Fast Fourier Transform**, runs in **nearly linear** time $O(\ln(n) \cdot \ln(\ln(n)) \cdot \ln(\ln(\ln(n)))) \subseteq O(\ln^{1+\epsilon}(n))$, for all $\epsilon > 0$.

2 Divisibility

(2.1) Greatest common divisors. a) Let $a, b \in \mathbb{Z}$. Then a is called a **divisor** of b, and b is called a **multiple** of a, if there is $c \in \mathbb{Z}$ such that $ac = b$; we write $a \mid b$. An element $c \in \mathbb{Z}$ such that $c \mid a$ and $c \mid b$ is called a **common divisor** of a and b. A common divisor $d \in \mathbb{Z}$ of a and b such that $c \mid d$, whenever $c \in \mathbb{Z}$ is a common divisor of a and b, is called a **greatest common divisor** of a and b; we will show below that greatest common divisors always exist.

If $d, d' \in \mathbb{Z}$ are greatest common divisors of a and b, then we have $d \mid d' \mid d$, hence $d' \in \{\pm d\}$; we let $\gcd(a, b) \in \mathbb{N}_0 \subseteq \mathbb{Z}$ be the unique non-negative greatest common divisor of a and b. If $\gcd(a, b) = 1$ then a and b are called **coprime**.

b) Let $a \in \mathbb{Z} \setminus \{0, \pm 1\}$. Then a is called **indecomposable**, if $a = bc \in \mathbb{Z}$, where $b, c \in \mathbb{Z}$, implies $b \in \{\pm 1\}$ or $c \in \{\pm 1\}$; otherwise a is called **composite**.

Let $\mathcal{P} \subseteq \mathbb{N} \subseteq \mathbb{Z}$ be the set of non-negative indecomposable elements of $\mathbb{Z}$. Then any element $0 \neq a \in \mathbb{Z}$ has a unique **factorisation** $a = \epsilon_a \cdot \prod_{p \in \mathcal{P}} p^{\nu_p(a)}$, where $\epsilon_a \in \{\pm 1\}$, and $\nu_p(a) \in \mathbb{N}_0$ is called the associated **multiplicity**; we have $\nu_p(a) = 0$ for almost all $p \in \mathcal{P}$.

In particular, for $0 \neq a, b \in \mathbb{Z}$ we have $\gcd(a, b) = \prod_{p \in \mathcal{P}} p^{\min\{\nu_p(a), \nu_p(b)\}}$, in particular showing that greatest common divisors always exist.

c) Let $a \in \mathbb{Z} \setminus \{0, \pm 1\}$. Then a is called a **prime**, if $a \mid bc$ implies $a \mid b$ or $a \mid c$, where $b, c \in \mathbb{Z}$. We show that a is a prime if and only if a is indecomposable:

Firstly, let a be a prime, and let $a = bc$ for some $b, c \in \mathbb{Z}$. Hence we may assume that $a \mid b$, thus there is $d \in \mathbb{Z}$ such that $ad = b$, hence $a = adc$, which since $a \neq 0$ implies $cd = 1$, yielding $c \in \{\pm 1\}$. Secondly, let a be indecomposable, and let $b, c \in \mathbb{Z}$ such that $a \mid bc$. Hence there is $d \in \mathbb{Z}$ such that $ad = bc = \epsilon_b \epsilon_c \cdot \prod_{p \in \mathcal{P}} p^{\nu_p(b)+\nu_p(c)}$. Since a is indecomposable, uniqueness of factorisation implies $a \in \{\pm p\}$, for some $p \in \mathcal{P}$ such that $\nu_p(b) + \nu_p(c) > 0$. Hence we have $\nu_p(b) > 0$ or $\nu_p(c) > 0$, implying $a \mid b$ or $a \mid c$. ♯

(2.2) Extended Euclidean algorithm Let $a, b \in \mathbb{N}$. Then the greatest common divisor $r = \gcd(a, b) \in \mathbb{N}$, and **Bézout coefficients** $s, t \in \mathbb{Z}$ such that $r = sa + tb \in \mathbb{Z}$, can be computed as follows, without determining the factorisation of a and b. Leaving out the steps indicated by $\circ$, only needed to compute s and t, just yields the greatest common divisor r, the remaining algorithm being called the **Euclidean algorithm**.

First recall that $\mathbb{Z}$ allows for **quotient and remainder**, saying that for any $m, n \in \mathbb{Z}$, such that $n \neq 0$, there are uniquely determined $q \in \mathbb{Z}$ and $r \in \{0, \ldots, |n| - 1\}$ such that $m = qn + r \in \mathbb{Z}$. We do not discuss the question how q and r can be determined algorithmically, but just mention that this again can be done using $O(\ln(m)\ln(n))$ bit operations.

- $r_0 \leftarrow a, \quad r_1 \leftarrow b$
- $\circ$ $s_0 \leftarrow 1, \quad t_0 \leftarrow 0$
- $\circ$ $s_1 \leftarrow 0, \quad t_1 \leftarrow 1$
- $i \leftarrow 1$
- while $r_i > 0$ do
  - $r_{i+1} \leftarrow r_{i-1} \bmod r_i$ # quotient and remainder $r_{i-1} = q_i r_i + r_{i+1}$
  - $q_i \leftarrow \lfloor \frac{r_{i-1}}{r_i} \rfloor$
  - $\circ$ $s_{i+1} \leftarrow s_{i-1} - q_i s_i$
  - $\circ$ $t_{i+1} \leftarrow t_{i-1} - q_i t_i$
  - $i \leftarrow i + 1$
- return $[r; s, t] \leftarrow [r_{i-1}; s_{i-1}, t_{i-1}]$ # resp. $r \leftarrow r_{i-1}$

We have $r_0 = s_0 a + t_0 b$ and $r_1 = s_1 a + t_1 b$, and by induction on $i \geq 1$ we have $r_{i+1} = r_{i-1} - q_i r_i = (s_{i-1}a + t_{i-1}b) - q_i \cdot (s_i a + t_i b) = s_{i+1}a + t_{i+1}b$. As we have $r_i < r_{i-1}$ for all $i \geq 1$, there is $l \in \mathbb{N}_0$ such that $r_l > 0$ and $r_{l+1} = 0$. Hence the algorithm terminates, after l executions of the while loop, returning $[r, s, t] := [r_l, s_l, t_l]$ such that $r = sa + tb$. Moreover, from $r_{i+1} = r_{i-1} - q_i r_i$, for all $i \in \{1, \ldots, l\}$, we indeed get $r = r_l = \gcd(r_l, 0) = \gcd(r_l, r_{l+1}) = \gcd(r_i, r_{i+1}) = \gcd(r_{i-1}, r_i) = \gcd(r_0, r_1) = \gcd(a, b)$.

The input lengths of the integers occurring in the Euclidean algorithm are bounded by the input lengths of a and b, and since the number l of executions of loop can be bounded by a polynomial in the input lengths of a and b, we conclude that the Euclidean algorithm is a polynomial-time algorithm.

(2.3) Trial division and Sieve of Erathostenes. For $x \in \mathbb{R}$ let $\mathcal{P}_{\leq x} := \{p \in \mathcal{P}; p \leq x\}$, and let $\pi(x) := |\mathcal{P}_{\leq x}| \in \mathbb{N}_0$; the asymptotic behaviour of $\pi(x)$ is

described by the Prime Number Theorem saying that $\pi(x) \sim \frac{x}{\ln(x)}$.

a) The straightforward approach to compute the factorisation of $1 \neq n \in \mathbb{N}$ is to use **trial division** with respect to $\mathcal{P}_{\leq \sqrt{n}}$; since any proper prime divisor of n is contained in $\mathcal{P}_{\leq \sqrt{n}}$, if n is prime then this also proves primality. Note that by the Prime Number Theorem trial division an exponential-time algorithm with running time $O(\exp((\frac{1}{2} + o(1)) \ln(n)))$.

- $L \leftarrow []$
- for $p \in \mathcal{P}_{\leq \sqrt{n}}$ do
  - while $(n \bmod p) = 0$ do
    - append(L,p)
    - $n \leftarrow \frac{n}{p}$
- if $n = 1$ then return L # n decomposable, factorisation found
- else return $[n]$ # n is a prime

b) Hence we need an algorithm to compute the sets $\mathcal{P}_{\leq n}$, for $1 \neq n \in \mathbb{N}$. This is achieved by the **Sieve of Erathostenes**:

- $L \leftarrow [2, \ldots, n]$
- $k \leftarrow 1$
- while $k < n$ do
  - $k \leftarrow k + 1$
  - if k in L then
    - $j \leftarrow 2k$
    - while $j \leq n$ do
      - delete(L,j)
      - $j \leftarrow j + k$
- return L

3 Modular arithmetic

(3.1) Groups. a) A set G together with a **multiplication** $\cdot\colon G \times G \to G$ fulfilling the following conditions is called a **commutative group**: We have **commutativity** $ab = ba$ for all $a, b \in G$; we have **associativity** $(ab)c = a(bc)$ for all $a, b, c \in G$; there is a **neutral element** $1 \in G$ such that $a \cdot 1 = a$ for all $a \in G$; and for any $a \in G$ there is an **inverse** $a^{-1} \in G$ such that $a \cdot a^{-1} = 1$.

It then follows immediately that $1 \in G$ is the unique neutral element, and that $a^{-1} \in G$ is the unique inverse of $a \in G$.

b) If G is finite, then the cardinality $|G| \in \mathbb{N}$ is called the **order** of G. Then by **Lagrange's Theorem** we have $a^{|G|} = 1 \in G$, for all $a \in G$.

Moreover, the **order** of $a \in G$ is defined as $\mathrm{ord}_G(a) := \min\{n \in \mathbb{N}; a^n = 1\} \in \mathbb{N}$; note that by Lagrange's Theorem this is well-defined. Then we have $a^m = 1$, for $m \in \mathbb{Z}$, if and only if $\mathrm{ord}_G(a) \mid m$; note that hence in particular $\mathrm{ord}_G(a) \mid |G|$:

Let $n := \mathrm{ord}_G(a)$. If $m = kn$, for some $k \in \mathbb{Z}$, then we have $a^m = (a^n)^k = 1$. Conversely, if $a^m = 1$, then by quotient and remainder writing $m = qn + r$,

where $q \in \mathbb{Z}$ and $r \in \{0, \dots, n-1\}$, yields $a^r = a^{m-qn} = a^m \cdot (a^n)^q = 1$, thus by the minimality of n we conclude $r = 0$, that is $n \mid m$. ♯

(3.2) Modular arithmetic. a) For $n \geq 2$ let $\mathbb{Z}_n := \{0, \dots, n-1\} \subseteq \mathbb{Z}$, and for $a \in \mathbb{Z}$ let $\overline{a} := (a \bmod n) \in \mathbb{Z}_n$ be the remainder of a upon division by n. We define an addition $+\colon \mathbb{Z}_n \times \mathbb{Z}_n \to \mathbb{Z}_n$ and a multiplication $\cdot\colon \mathbb{Z}_n \times \mathbb{Z}_n \to \mathbb{Z}_n$ by $a + b := \overline{a+b}$ and $a \cdot b := \overline{ab}$.

We have independence from the choice of representatives: If $a, a', b, b' \in \mathbb{Z}$ such that $\overline{a} = \overline{a'}$ and $\overline{b} = \overline{b'}$, then we have $\overline{a+b} = \overline{a'+b'}$ and $\overline{ab} = \overline{a'b'}$. This shows that the laws of commutativity, associativity and distributivity are inherited from $\mathbb{Z}$. Moreover, iterated arithmetic operations can be performed in $\mathbb{Z}$ before going over to remainders with respect to n, although this is usually not recommended.

As for addition, we have a **neutral element** $0 \in \mathbb{Z}_n$, and any $a \in \mathbb{Z}_n$ has an additive **inverse** $\overline{-a} \in \mathbb{Z}_n$; hence $\mathbb{Z}_n$ is a commutative additive group. As for multiplication, we have a neutral element $1 \in \mathbb{Z}_n$, but a multiplicative inverse does not always exist:

b) An element $a \in \mathbb{Z}_n$ is called **invertible**, if there is $b \in \mathbb{Z}_n$ such that $ab = 1 \in \mathbb{Z}_n$. Let $\mathbb{Z}_n^* := \{a \in \mathbb{Z}_n; a \text{ invertible}\}$, then $\mathbb{Z}_n^*$ is a commutative multiplicative group, called the **group of units** of $\mathbb{Z}_n$; we have $1 \in \mathbb{Z}_n^*$ and $0 \notin \mathbb{Z}_n^*$. Then we have $\mathbb{Z}_n^* = \{a \in \mathbb{Z}_n; \gcd(a, n) = 1\}$; hence $\mathbb{Z}_n^*$ is also called the **group of prime residues** modulo n:

Note first that, whenever $a, a' \in \mathbb{Z}$ such that $\overline{a} = \overline{a'}$, then we have $\gcd(a, n) = \gcd(a', n)$. If $a \in \mathbb{Z}_n^*$, having an inverse $b \in \mathbb{Z}_n^*$, then $\overline{ab} = \overline{1} \in \mathbb{Z}_n$ says that there is $k \in \mathbb{Z}$ such that $ab + kn = 1 \in \mathbb{Z}$, implying that $\gcd(a, n) = 1$. Conversely, if $a \in \mathbb{Z}_n$ such that $\gcd(a, n) = 1$, then there are Bézout coefficients $s, t \in \mathbb{Z}$ such that $sa + tn = 1 \in \mathbb{Z}$, hence we have $\overline{1} = \overline{sa+tn} = \overline{sa} \in \mathbb{Z}_n$, saying that $\overline{s} \in \mathbb{Z}_n$ is an inverse of $\overline{a} \in \mathbb{Z}_n$. ♯

c) Let $\varphi\colon \mathbb{N} \to \mathbb{N}\colon n \mapsto |\mathbb{Z}_n^*|$ be **Euler's totient function**, where for completeness we let $\mathbb{Z}_1^* := \mathbb{Z}_1 = \{0\}$, hence $\varphi(1) := 1$.

For example, if $n := p^e$, where $p \in \mathbb{N}$ is a prime and $e \in \mathbb{N}$, then $\mathbb{Z}_n \setminus \mathbb{Z}_n^* = \{a \in \mathbb{Z}_n; \gcd(a, n) > 1\} = \{a \in \mathbb{Z}_n; p \mid a\} = \{kp \in \mathbb{Z}_n; k \in \{0, \dots, p^{e-1} - 1\}\}$, thus $\varphi(p^e) = p^e - p^{e-1} = p^{e-1}(p-1)$; in particular, $\varphi(p) = p - 1$.

If $n := pq$, where $p \neq q \in \mathbb{N}$ are primes, then $\mathbb{Z}_n \setminus \mathbb{Z}_n^* = \{a \in \mathbb{Z}_n; \gcd(a, n) > 1\} = \{a \in \mathbb{Z}_n; p \mid a\} \cup \{a \in \mathbb{Z}_n; q \mid a\} = \{kp \in \mathbb{Z}_n; k \in \{0, \dots, q-1\}\} \cup \{kq \in \mathbb{Z}_n; k \in \{0, \dots, p-1\}\}$, thus $\varphi(pq) = n - q - p + 1 = (p-1)(q-1)$.

(3.3) Modular exponentiation Let $n \geq 2$, let $a \in \mathbb{Z}_n$ and let $e \in \mathbb{N}$. Then the e-th power $b := a^e \in \mathbb{Z}_n$ can be computed by **repeated squaring** as follows: To this end, let $e = \sum_{j=0}^{k-1} e_j \cdot 2^j$ be the binary representation of e, where $e_i \in \{0, 1\}$ and $k := b_2(e)$.

• $b_0 \leftarrow 1, \quad a_0 \leftarrow a$

- for $i \in [0, \ldots, k-1]$ do
  - $b_{i+1} \leftarrow b_i$
  - if $e_i = 1$ then $b_{i+1} \leftarrow a_i b_{i+1} \bmod n$
  - $a_{i+1} \leftarrow a_i^2 \bmod n$
- return b_k

By induction on $i \in \{0, \ldots, k\}$ we have $a_i = a^{2^i} \in \mathbb{Z}_n$. Thus by induction on $i \in \{-1, \ldots, k-1\}$ we get $b_{i+1} = a^{\sum_{j=0}^{i} e_j 2^j}$: The case $i = -1$ being clear, let $i \geq 0$; if $e_i = 0$ then $b_{i+1} = b_i = a^{\sum_{j=0}^{i-1} e_j 2^j} = a^{\sum_{j=0}^{i} e_j 2^j}$, if $e_i = 1$ then $b_{i+1} = a_i b_i = a^{2^i + \sum_{j=0}^{i-1} e_j 2^j} = a^{\sum_{j=0}^{i} e_j 2^j} \in \mathbb{Z}_n$. This yields $b_k = a^{\sum_{j=0}^{k-1} e_j 2^j} = a^e \in \mathbb{Z}_n$.

All integers occurring are bounded above by n^2, hence have bit lengths in $O(\ln(n))$, and the number of multiplications needed is in $O(\ln(e))$. Hence this is a polynomial-time algorithm with running time in $O(\ln^2(n) \cdot \ln(e))$. Note that 'classical' exponentiation by computing $a^e \in \mathbb{Z}$ first produces integers of bit lengths in $O(e \cdot \ln(n))$, and even by taking remainders modulo n needs $O(e)$ multiplications; hence these are exponential-time algorithms.

4 The RSA cryptosystem

(4.1) Cryptosystems. A **cryptosystem** $[\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D}]$ is a tuple, where the **plaintexts** $\mathcal{P}$, the **ciphertexts** $\mathcal{C}$ and the **keys** $\mathcal{K}$ are finite sets, and where $\mathcal{E} = \{E_e \colon \mathcal{P} \to \mathcal{C}; e \in \mathcal{K}\}$ and $\mathcal{D} = \{D_d \colon \mathcal{C} \to \mathcal{P}; d \in \mathcal{K}\}$ are **encryption** and **decryption functions**, respectively, such that for all $e \in \mathcal{K}$ there is $d \in \mathcal{K}$ such that $D_d \circ E_e = \mathrm{id}_{\mathcal{P}}$.

To encrypt and decrypt plain texts, Latin letters are first **encoded** into and **decoded** from $\mathbb{Z}_{26} := \{0, \ldots, 25\}$, say, or electronic text data, using the ASCII alphabet of length 128, is encoded into and decoded from $\mathbb{Z}_{128} := \{0, \ldots, 127\}$. Hence a **word** of length $l \in \mathbb{N}$ is identified with an element in $\mathbb{Z}_z^l$, for some $z \geq 2$, and the latter is interpreted as the representation of a non-negative integer with respect to the base z. So, typically we have $\mathcal{P}, \mathcal{C} \subseteq \mathbb{Z}_z^l$.

The idea now is to keep information private to communication partners, Alice and Bob say, who communicate through an insecure channel, where data might be caught by an opponent, Oscar say. Hence plaintexts are first encrypted by Bob, then sent through the channel, and are decrypted again by Alice. Thus in practice, given the keys, encryption and decryption functions should be efficiently computable. Moreover, it should be difficult for Oscar to determine plaintexts from ciphertexts without knowing the keys used, and it should also be difficult for Oscar to determine the keys employed.

If given an encryption key $e \in \mathcal{K}$ a suitable decryption key $d \in \mathcal{K}$ can be assumed to be equal to e, or if d can be easily computed from e, then the cryptosystem is called **symmetric** or a **private-key cryptosystem**. In this case Alice and Bob first have to exchange the keys securely.

If a suitable decryption key $d \in \mathcal{K}$ cannot be computed easily from e, then the

cryptosystem is called **asymmetric** and can be used as a **public-key cryptosystem**: To receive messages Alice publishes $e \in \mathcal{K}$, which Bob uses to encrypt messages, but Alice keeps the suitable decryption keys $d \in \mathcal{K}$ private. In this case no secure key exchange is necessary.

(4.2) The Rivest-Shamir-Adleman (RSA) cryptosystem [1978]. a) Let $p \neq q \in \mathbb{N}$ be primes and let $n := pq \in \mathbb{N}$ be the associated **modulus**. Let $\mathcal{P} = \mathcal{C} := \mathbb{Z}_n^*$ and $\mathcal{K} := \mathbb{Z}_{\varphi(n)}^*$, where $\varphi(n) = (p-1)(q-1)$, Moreover, for $e \in \mathbb{Z}_{\varphi(n)}^*$ let $E_e \colon \mathbb{Z}_n^* \to \mathbb{Z}_n^* \colon a \mapsto a^e$, and let $\mathcal{E} = \mathcal{D} := \{E_e \colon \mathbb{Z}_n^* \to \mathbb{Z}_n^*; e \in \mathbb{Z}_{\varphi(n)}^*\}$. This indeed is an (unsymmetric) cryptosystem:

Given $e \in \mathbb{Z}_{\varphi(n)}^*$, we have to provide $d \in \mathbb{Z}_{\varphi(n)}^*$ such that $E_d \circ E_e = \mathrm{id}_{\mathbb{Z}_n^*}$. In order to do so, let $d \in \mathbb{Z}_{\varphi(n)}^*$ such that $ed = 1 \in \mathbb{Z}_{\varphi(n)}^*$. Then we have $ed = 1 + k\varphi(n) \in \mathbb{Z}$, for some $k \in \mathbb{Z}$, and since by Lagrange's Theorem we have $a^{\varphi(n)} = 1 \in \mathbb{Z}_n^*$, we get $(a^e)^d = a^{ed} = a \cdot (a^{\varphi(n)})^k = a \in \mathbb{Z}_n^*$, for all $a \in \mathbb{Z}_n^*$, as desired. Hence the **public key** is $[n, e]$, and the **private key** is $[p, q, d]$.

For example, letting $p := 97$ and $q := 193$, we get $n = 18721$, and letting $e := 43$, using $\varphi(n) = (p-1)(q-1) = 96 \cdot 192 = 18432$, the Euclidean algorithm yields $1 = \gcd(e, \varphi(n)) = -8573 \cdot e + 20 \cdot \varphi(n)$, hence we let $d := \overline{-8573} = 9859 \in \mathbb{Z}_{\varphi(n)}^*$.

(4.3) The RSA cryptosystem and integer factorisation. Let $p \neq q \in \mathbb{N}$ be primes and let $n := pq \in \mathbb{N}$. If $\varphi(n)$ is known, then inverses in $\mathbb{Z}_{\varphi(n)}^*$ can be computed, by the Euclidean algorithm, in polynomial time.

Computing $\varphi(n) = (p-1)(q-1)$ is **polynomial-time equivalent** to factoring $n = pq$: If p and q are known, then $\varphi(n) = (p-1)(q-1)$ can be computed as well. Conversely, if $\varphi(n) = (p-1)(q-1) = (p-1)(\frac{n}{p} - 1)$ is known, then $p^2 + (\varphi(n) - n - 1)p + n = 0$ shows that $\{p, q\}$ can be determined as the roots of a quadratic equation.

In particular, for $0 \neq a \in \mathbb{Z}_n \setminus \mathbb{Z}_n^*$ we have $1 < \gcd(a, n) < n$, thus we have found a prime divisor of n. Thus these elements have to be excluded, but since $\frac{n-1-\varphi(n)}{n} = \frac{p+q}{pq} = \frac{1}{p} + \frac{1}{q}$, they are extremely rare anyway.

Hence breaking the RSA cryptosystem **polynomial-time reduces** to factoring n, thus the RSA cryptosystem is secure only if factoring $n = pq$ is computationally difficult. It is conjectured that factoring integers of the form pq is as difficult as factoring arbitrary integers, and that integer multiplication is a **(cryptographic) one-way function**. Conversely it is conjectured that factoring $n = pq$ polynomial-time reduces to breaking the RSA cryptosystem, which would imply that these problems are polynomial-time equivalent.

Given the capabilities of the known factorisation algorithms, p and q should be chosen of the same size, which should be at least $p \sim 2^{512} \sim 10^{154}$, thus $n \sim 2^{1024} \sim 10^{308}$. If there are factorisation algorithms running faster for certain choices of the prime divisors of n, for example if $p - 1$ or $p + 1$ has no large prime divisors, then these have to be avoided as well; and $|p - q|$ must not be

too small, since otherwise the prime divisors of n can be found by trial division with integers close to $\sqrt{n}$.

5 Primality testing

(5.1) Deterministic primality testing. Let $1 \neq n \in \mathbb{N}$. Then n is a prime if and only if there is a **primitive root** $a \in \mathbb{Z}_n^*$ such that $\operatorname{ord}(a) = n - 1$:

The number n is a prime if and only if $\varphi(n) = n - 1$. Thus, if n is a prime then by Lagrange's Theorem we have $\operatorname{ord}(a) \mid n - 1$ for all $a \in \mathbb{Z}_n^*$, and by **Artin's Theorem** $\mathbb{Z}_n^*$ is **cyclic**, that is there is an element $a \in \mathbb{Z}_n^*$ such that $\operatorname{ord}(a) = n-1$. Conversely, if there is an element $a \in \mathbb{Z}_n^*$ such that $\operatorname{ord}(a) = n-1$ then $n - 1 \mid \varphi(n)$, hence we have $\varphi(n) = n - 1$. ♯

To verify primality we have the **Lucas test [1876]**: Let $n-1 = \prod_{i=1}^{r} p_i^{e_i}$, where $r \in \mathbb{N}_0$, the $p_1, \ldots, p_r \in \mathbb{N}$ are pairwise distinct primes and $e_1, \ldots, e_r \in \mathbb{N}$. Then $a \in \mathbb{Z}_n^*$ has order $n - 1$ if and only if $a^{n-1} = 1 \in \mathbb{Z}_n^*$ and $a^{\frac{n-1}{p_i}} \neq 1 \in \mathbb{Z}_n^*$, for all $i \in \{1, \ldots, r\}$: The conditions imply that $\operatorname{ord}(a) \mid n - 1$; and assuming that $\operatorname{ord}(a) < n - 1$, we conclude that $\operatorname{ord}(a)$ divides a maximal proper divisor of $n - 1$, contradicting the conditions.

If n is a prime, then the tuple $[a; p_1, \ldots, p_r]$, where $a \in \mathbb{Z}_n^*$ is a primitive root, is a called a **Lucas primality witness** for n. Note that to do this we need a factorisation algorithm to find factors of $n - 1$, and we might have to apply the Lucas test recursively to verify primality of the candidate prime factors found.

(5.2) Probabilistic primality testing. Let $1 \neq n \in \mathbb{N}$. To verify primality only with a certain probability we have the **Fermat test**: If there is $a \in \mathbb{Z}_n^*$ such that $a^{n-1} \neq 1 \in \mathbb{Z}_n^*$, then n is composite, and a is called a **Fermat compositeness witness** for n.

If n is composite and $a^{n-1} = 1 \in \mathbb{Z}_n^*$ for some $1 \neq a \in \mathbb{Z}_n^*$, then n is called a **Fermat pseudo-prime** with respect to the **base** a, and a is called a **Fermat liar** for n. If n is a Fermat pseudo-prime with respect to all bases $a \neq x \in \mathbb{Z}_n^*$, then n is called a **Carmichael number [1910]**. It can be shown that there are infinitely many Carmichael numbers, where those up to 10^4 are

$$561, 1105, 1729, 2465, 2821, 6601, 8911.$$

The set $U_n := \{a \in \mathbb{Z}_n^*; a^{n-1} = 1\} \leq \mathbb{Z}_n^*$ is a **subgroup**, that is closed with respect to taking inverses and products. We have $U_n = \mathbb{Z}_n^*$ if and only if n is either a prime or a Carmichael number. If $U_n \neq \mathbb{Z}_n^*$ then we have $\frac{|U_n|}{|\mathbb{Z}_n^*|} \leq \frac{1}{2}$, implying that the fraction of compositeness witnesses is at least $\frac{1}{2}$. This leads to the following polynomial-time **Monte-Carlo algorithm**:

Given an **error bound** $0 < \epsilon < 1$, for at least $\lceil -\log_2(\epsilon) \rceil$ randomly chosen elements of $\mathbb{Z}_n^* \setminus \{1\}$ we perform the Fermat test; if a compositeness witness is found then composite is returned, otherwise probably prime is returned.

Thus, if n is prime then the answer is correct, but if n is composite then the answer is incorrect with a probability of at most ϵ; in other words, if the answer is composite then it correct, but if the answer is probably prime then it is incorrect with a probability of most ϵ.

(5.3) Probabilistic primality testing II. The workhorse of modern primality testing is the **Miller-Rabin test**, again a polynomial-time Monte-Carlo algorithm, based on the following observation [Miller, 1976; Rabin, 1980]:

Let $1 \neq n \in \mathbb{N}$ be odd, let $n-1 = 2^l m$, where $l \in \mathbb{N}$ and $m \in \mathbb{N}$ is odd, and let $B_n := \{a \in \mathbb{Z}_n^*; a^m = 1\} \dot\cup \coprod_{k=0}^{l-1} \{a \in \mathbb{Z}_n^*; a^{2^k m} = -1\} \subseteq U_n := \{a \in \mathbb{Z}_n^*; a^{n-1} = 1\}$. Then the following holds: If n is a prime then we have $B_n = \mathbb{Z}_n^*$, while if $n \neq 9$ is composite then we have $\frac{|B_n|}{|\mathbb{Z}_n^*|} \leq \frac{1}{4}$; note that for $n = 9$ we have $B_9 = \{\pm 1\}$ and $\varphi(9) = 6$.

An element $a \in \mathbb{Z}_n^* \setminus B_n$ is called a **strong compositeness witness** for n, hence for composite $n \neq 9$ the fraction of compositeness witnesses is at least $\frac{3}{4}$; note that in particular there are no 'strong Carmichael numbers'. If n is composite and $a \in B_n$, then n is called a **strong pseudo-prime** with respect to the **base** a, and a is called a **strong liar** for n; in this case we have $a^{n-1} = x^{2^l m} = 1 \in \mathbb{Z}_n^*$, hence a also is a Fermat liar.

Although there are composite $n \neq 9$ which are strong pseudo-primes with respect to a fraction of $\frac{1}{4}$ of the bases, for most n this fraction is much smaller. Moreover, although it is possible to construct strong pseudo-primes for any given finite set of bases, these are extremely rare.

6 Factorisation

(6.1) The ρ-method [Pollard, 1975]. Let $1 \neq n \in \mathbb{N}$, let $x_0 \in \mathbb{Z}_n$, and given a function $f\colon \mathbb{Z}_n \to \mathbb{Z}_n$ let recursively $x_i := f(x_{i-1})$, for $i \in \mathbb{N}$. We assume that f is chosen such that the x_i behave like independent random choices in $\mathbb{Z}_n$. In practice, functions $f_c\colon x \mapsto x^2 + c$, for $c \in \mathbb{Z}_n$, are used, where a typical choice is $c \in \{\pm 1\}$; note that $f_0(x) = x^2$, and that $f_{-2}(x + x^{-1}) = x^2 + x^{-2}$ for $x \in \mathbb{Z}_n^*$, hence f_0 and f_{-2} do not behave randomly.

Let $1 \neq p \in \mathbb{N}$ be a divisor of n. Then there are $k \in \mathbb{N}_0$ and $l \in \mathbb{N}$ minimal such that we have a **collision** $x_k = x_{k+l} \in \mathbb{Z}_p$; hence we have $x_k = x_{k+jl} \in \mathbb{Z}_p$, for all $j \in \mathbb{N}_0$, thus the name of the method. Hence we have $p \mid \gcd(x_k - x_{k+l}, n)$.

To find a collision, it can be avoided to store the sequence $[x_0, x_1, \ldots]$ by using **Floyd's cycle detection trick**: Let $y_0 := x_0 \in \mathbb{Z}_n$ and $y_i := f(f(y_{i-1})) = x_{2i} \in \mathbb{Z}_n$, for $i \in \mathbb{N}$. Then we have $x_i = y_i \in \mathbb{Z}_p$ if and only if $i \geq k$ and $l \mid 2i - i = i$. Thus the minimal $i \in \mathbb{N}$ fulfilling these conditions is an element of $\{k, \ldots, k + l\}$, hence we still need at most $k + l$ steps to arrive at a collision.

The number of steps needed, until we arrive at a collision with a given probability $1 - \epsilon$, where $0 < \epsilon < 1$, is of size $O(\sqrt{p}) \subseteq O(\sqrt[4]{n})$: For $t \in \mathbb{N}_0$, precisely

$\prod_{i=0}^{t}(p-i)$ of the sequences in $\mathbb{Z}_p^{t+1}$ have pairwise distinct entries. By Taylor series expansion, for $0 \le \lambda \le 1$ we have $0 \le \exp(-\lambda)-(1-\lambda) \le \frac{\lambda^2}{2}$, hence for the fraction of the sequences with pairwise distinct entries amongst all sequences we get $\prod_{i=0}^{t}(1-\frac{i}{p}) \le \prod_{i=0}^{t}\exp(-\frac{i}{p}) = \exp(\frac{-t(t+1)}{2p}) \le \exp(\frac{-t^2}{2p})$, where $\exp(\frac{-t^2}{2p}) < \epsilon$ if and only if $t > \sqrt{-2p \cdot \ln(\epsilon)} \in O(\sqrt{p})$. Note that for $\epsilon = \frac{1}{2}$ and $p = 365$ this yields $t \ge 23$, being called the **birthday paradox**.

Let $q := \frac{n}{p} \in \mathbb{N}$, and assume that $q \neq 1$ and $\gcd(p,q) = 1$; note that p and q exist if and only if n is not a prime power. Then the natural map $\mathbb{Z}_n = \mathbb{Z}_{pq} \to \mathbb{Z}_p \times \mathbb{Z}_q \colon x \mapsto [x \bmod p, x \bmod q]$ is a bijection: Letting $a, b \in \mathbb{Z}$ such that $1 = ap+bq \in \mathbb{Z}$, we have $bq = 1 \in \mathbb{Z}_p$ and $bq = 0 \in \mathbb{Z}_q$, as well as $ap = 0 \in \mathbb{Z}_p$ and $ap = 1 \in \mathbb{Z}_q$, by additivity showing surjectivity, thus bijectivity. Thus we infer that $x_i \in \mathbb{Z}_p$ and $x_i \in \mathbb{Z}_q$ can be considered as independent random choices in $\mathbb{Z}_p$ and $\mathbb{Z}_q$, respectively. Hence with probability $\frac{q-1}{q}$ we have $q \not| \gcd(x_k - x_{k+l}, n)$, implying that $1 < \gcd(x_k - x_{k+l}, n) < n$, yielding a proper divisor of n.

This yields the following exponential-time **Las-Vegas algorithm** to factor n, which is assumed not to be a prime power, where $f \colon \mathbb{Z}_n \to \mathbb{Z}_n$, and $m \in \mathbb{N}$ is the maximum number of tries made, chosen as $m \sim \sqrt[4]{n} = \exp(\frac{1}{4} \cdot \ln(n))$.

- choose $x \in \mathbb{Z}_n$ randomly
- $y \leftarrow x$
- $i \leftarrow 0$
- while $i < m$ do
  - $i \leftarrow i + 1$
  - $x \leftarrow f(x)$
  - $y \leftarrow f(f(y))$
  - $g \leftarrow \gcd(x - y, n)$
  - if $1 < g < n$ then return g
- return fail

For example, for the Fermat number $F_5 := 2^{2^5}+1 = 4\,294\,967\,297 \sim 4 \cdot 10^9$ the ρ-method with $c := 1$ and $x_0 := 1$ needs 22 tries to find the divisor $p_3 := 641 \mid F_5$, hence $p_7 := \frac{F_5}{p_3} = 6\,700\,417$ [Euler, 1732]. From $p_3 - 1 = 2^7 \cdot 5$ we find the Lucas witness $[3; 2, 5]$ for p_3; from $p_7 - 1 = 2^7 \cdot 3 \cdot 17449$ and $17449 - 1 = 2^3 \cdot 3 \cdot 727$, we find the Lucas witnesses $[5; 2, 3, 17449]$ for p_7, and $[17; 2, 3, 727]$ for 17449; here we take the primes up to 1000 for granted.

For the Fermat number $F_6 := 2^{2^6} + 1 \sim 1.8 \cdot 10^{19}$, the ρ-method with $c := 1$ and $x_0 := 1$ needs 808 tries to find the divisor $p_6 := 274177 \mid F_6$, hence $p_{14} := \frac{F_6}{p_6} = 67\,280\,421\,310\,721$ [Landry, 1880]. From $p_6 - 1 = 2^8 \cdot 3^2 \cdot 7 \cdot 17$ we find the Lucas witness $[5; 2, 3, 7, 17]$ for p_6; from $p_{14} - 1 = 2^8 \cdot 5 \cdot 47 \cdot 373 \cdot 2\,998\,279$ and $2\,998\,279 - 1 = 2 \cdot 3^2 \cdot 166571$ and $166571 - 1 = 2 \cdot 5 \cdot 16657$ and $16657 - 1 = 2^4 \cdot 3 \cdot 347$, we find the Lucas witnesses $[3; 2, 5, 47, 373, 2\,998\,279]$ for p_{14}, and $[3; 2, 3, 166571]$ for $2\,998\,279$, and $[2; 2, 5, 16657]$ for 166571, and $[5; 2, 3, 347]$ for 16657.

For the Fermat number $F_9 := 2^{2^9} + 1 \sim 1.3 \cdot 10^{154}$, the ρ-method with $c := 1$ and $x_0 := 1$ needs 1563 tries to find the divisor $p_7 := 2\,424\,833 \mid F_9$. From

$p_7 - 1 = 2^{16} \cdot 37$ we find the Lucas witness $[3; 2, 37]$. For $n := \frac{F_6}{p_6} \sim 5.5 \cdot 10^{146}$ the Fermat test yields the compositeness witness 3; indeed we have $n = p_{49} \cdot p_{99}$, the prime factors having the indicated number of digits [Lenstra, 1990].

(6.2) The $(p-1)$-method [Pollard, 1974]. An integer $x \in \mathbb{N}$ is called **b-smooth**, where $b \in \mathbb{N}$, if $x = \prod_{q \in \mathcal{P}_{\leq b}} q^{\nu_q(x)}$, that is the prime divisors of x do not exceed b.

Let $1 \neq n \in \mathbb{N}$, let $b \leq n$, and let $p \in \mathbb{N}$ be a prime divisor of n such that $p-1$ is b-smooth. Then, for $x \in \mathbb{Z}_n^*$ and $e \in \mathbb{N}$ such that $p - 1 \mid e$ we have $x^e = 1 \in \mathbb{Z}_p^*$, thus $\gcd(x^e - 1, n) > 1$. To find a suitable exponent e, we observe that $\nu_q(p-1) \leq \log_q(p) \leq \log_q(n)$, for all $q \in \mathcal{P}_{\leq b}$, hence by smoothness we may choose $e := \operatorname{lcm}\left(m^{\lfloor \log_m(n) \rfloor} \in \mathbb{N}; m \in \{2, \ldots, b\}\right) \in \mathbb{N}$.

Letting $n = \prod_{q \in \mathcal{P}} q^{e_q}$, the natural map $\mathbb{Z}_n \to \prod_{q \in \mathcal{P}, e_q > 0} \mathbb{Z}_{q^{e_q}}$ is a bijection, and we have a natural map $\mathbb{Z}_{q^{e_q}} \to \mathbb{Z}_q$, where by Artin's Theorem $\mathbb{Z}_q^*$ is cyclic. This shows that if n has a prime divisor $q \neq p$ such that $q-1$ is not b-smooth, then it is likely that $x^e \neq 1 \in \mathbb{Z}_q^*$, implying that $\gcd(x^e - 1, n) < n$; recall that we may assume that n is not a prime power. This yields the following Las-Vegas algorithm to factor n:

- choose $x \in \mathbb{Z}_n^*$ randomly
- $e \leftarrow 1$
- for $m \in [2, \ldots, b]$ do
  - $d \leftarrow \lfloor \log_m(n) \rfloor$
  - $e \leftarrow \mathsf{lcm}(e, m^d)$
- $y \leftarrow x^e \bmod n$
- $g \leftarrow \gcd(y - 1, n)$
- if $1 < g < n$ then return g
- else return fail

For example, for the Fermat number $F_5 := 2^{2^5} + 1 = 4\,294\,967\,297 \sim 4 \cdot 10^9$, using $b := 5$ and $x := 3$, we recover the prime divisor $p_3 := 641 \mid F_5$; for the Fermat number $F_6 := 2^{2^6} + 1 \sim 1.8 \cdot 10^{19}$, using $b := 17$ and $x := 3$, we recover the prime divisor $p_6 := 274177 \mid F_6$; for the Fermat number $F_9 := 2^{2^9} + 1 \sim 1.3 \cdot 10^{154}$, using $b := 37$ and $x := 3$, we recover the prime divisor $p_7 := 2\,424\,833 \mid F_9$.

(6.3) The random squares method [Dixon, 1981]. Let $1 \neq n \in \mathbb{N}$ be odd; recall that we may assume that n is not a prime power. Then we have mutually inverse bijections $\{[x, y] \in \mathbb{N}^2; x \geq y, n = xy\} \leftrightarrow \{[s, t] \in \mathbb{N}_0^2; s > t, n = s^2 - t^2\}$ given by $[x, y] \mapsto [\frac{x+y}{2}, \frac{x-y}{2}]$ and $[s, t] \mapsto [s + t, s - t]$. Hence finding a divisor of n is equivalent to writing n is a difference of two squares.

Hence, if $x, y \in \mathbb{Z}_n$ such that $y \notin \{\pm x\} \subseteq \mathbb{Z}_n$ and $x^2 = y^2 \in \mathbb{Z}_n$, then we have $(x + y)(x - y) = x^2 - y^2 = kn$ for some $k \in \mathbb{Z}$, and thus $\gcd(x + y, n) > 1$ or $\gcd(x - y, n) > 1$, where both $\gcd(x + y, n) < n$ and $\gcd(x - y, n) < n$. This yields the **Fermat-Legendre method** to factor n: We choose $k \in \mathbb{N}$ small, and for increasing $x \geq \lceil \sqrt{kn} \rceil$ we check whether $x^2 - kn \in \mathbb{N}$ is a square.

Letting $n = \prod_{q\in\mathcal{P}} q^{e_q}$, the natural map $\mathbb{Z}_n \to \prod_{q\in\mathcal{P},e_q>0} \mathbb{Z}_{q^{e_q}}$ is a bijection, and since for q odd the group $\mathbb{Z}_{q^{e_q}}^*$ is cyclic (a generalisation of Artin's Theorem), we conclude that any square in $\mathbb{Z}_n^*$ has precisely 2^r square roots, where $r := |\{q \in \mathcal{P}; e_q \geq 1\}| \in \mathbb{N}$ is the number of prime divisors of n. Thus, given $x \in \mathbb{Z}_n^*$, if $y \in \mathbb{Z}_n^*$ is randomly chosen such that $x^2 = y^2 \in \mathbb{Z}_n^*$, then with probability $\frac{2^r-2}{2^r} = 1 - \frac{1}{2^{r-1}}$ we have $y \notin \{\pm x\}$. Hence if $r \geq 2$ this yields a proper divisor of n with probability $\geq \frac{1}{2}$. Still, given $x \in \mathbb{Z}_n^*$, the task to find some $y \in \mathbb{Z}_n^* \setminus \{\pm x\}$ such that $x^2 = y^2 \in \mathbb{Z}_n^*$ remains:

Let $b \in \mathbb{N}$, and assume that $\mathcal{P}_{\leq b} = \{p_1, \ldots, p_l\}$, where $l \in \mathbb{N}_0$, is known. Letting additionally $p_0 := -1$, the sequence $[p_0, p_1, \ldots, p_l]$ is called the **factor base** associated with b. For $x \in \mathbb{Z}_n^*$ let $x' \in \{-\frac{n-1}{2}, \ldots, \frac{n-1}{2}\}$ such that $x^2 = x' \in \mathbb{Z}_n^*$. If $|x'| \in \mathbb{N}$ is b-smooth, then we have $x' = \prod_{i=0}^{l} p_i^{e_i(x)} \in \mathbb{Z}$, where $e_i(x) \in \mathbb{N}_0$ and $e_0(x) \leq 1$; in this case x is called a **b-number**, the sequence $e(x) := [e_0(x), \ldots, e_l(x)] \in \mathbb{N}_0^{l+1}$ is called its **exponent vector**, and going over to the field $\mathbb{Z}_2$ we get the **reduced** exponent vector $\epsilon(x) := [\epsilon_0(x), \ldots, \epsilon_l(x)] \in \mathbb{Z}_2^{l+1}$.

Let $x_1, \ldots, x_k \in \mathbb{Z}_n^*$ be b-numbers fulfilling the relation $\sum_{j=1}^{k} \epsilon(x_j) = 0 \in \mathbb{Z}_2^{l+1}$, thus $f_i := \sum_{j=1}^{k} e_i(x_j) \in \mathbb{N}$ is even, for all $i \in \{0, \ldots, l\}$. Letting $x := \prod_{j=1}^{k} x_j \in \mathbb{Z}$ and $y := \prod_{i=0}^{l} p_i^{\frac{f_i}{2}} \in \mathbb{Z}$ we have $y^2 = \prod_{i=0}^{l} p_i^{f_i} = \prod_{j=1}^{k} \prod_{i=0}^{l} p_i^{e_i(x_j)} = \prod_{j=1}^{k} x_j' \in \mathbb{Z}$, implying that $y^2 = x^2 \in \mathbb{Z}_n^*$. Note that exponent vectors can be computed independently from each other, which hence can be done using distributed computing. Moreover, a $\mathbb{Z}_2$-linear dependency as above occurs for some $k \leq l + 2$, and is found using Gaussian elimination.

This yields a Las-Vegas algorithm to factor n, where, using the **Canfield-Erdős-Pomerance Theorem [1983]** on the fraction of b-smooth integers in $\mathbb{Z}_n^*$, we have **Dixon's Theorem**: Letting $L(n) := \exp(\sqrt{\ln(n) \cdot \ln(\ln(n))})$, a proper divisor of n is found, using a factor base of size $l \sim \sqrt{L(n)}$ and needing $l^3 \sim L(n)^{\frac{3}{2}}$ tries, in expected **sub-exponential** time $O(L(n)^{2+o(1)})$.

(6.4) The quadratic sieve method [Pomerance, 1984]. We keep the above setting, and let $m := \lfloor \sqrt{n} \rfloor \in \mathbb{N}$ and $f := (X + m)^2 - n \in \mathbb{Z}[X]$. Rather than looking for b-numbers amongst random choices from $\mathbb{Z}_n^*$, it should be much more likely to find b-smooth integers amongst the $f(x) \in \mathbb{Z}$ whenever $x \in \mathbb{Z}$ is small. To this end, we choose $c \in \mathbb{N}$, small compared to $\sqrt{n}$, and consider the **sieve interval** $\mathcal{I} := \{-c, \ldots, c\}$:

For $x \in \mathcal{I}$ we have $f(x) = x^2 + 2xm + (m^2 - n) \sim 2xm$, hence $|f(x)|$ is bounded by $\sim 2c \cdot \sqrt{n} < \frac{n-1}{2}$. Then the **Pomerance Conjecture** says that the fraction of b-smooth integers in $\{f(x) \in \mathbb{Z}; x \in \mathcal{I}\}$ is asymptotically the same as the fraction of b-smooth integers in $\{1, \ldots, \lfloor \sqrt{n} \rfloor\}$; in practice, the expected number of tries necessary typically is much less than $\sim l^3$.

Moreover, for any odd prime $p \in \mathbb{N}$ such that $\gcd(p, n) = 1$, from $f(x) = 0 \in \mathbb{Z}_p$ we infer that $n = (x + m)^2 \in \mathbb{Z}_p^*$ is a square. Since by Artin's Theorem $\mathbb{Z}_p^*$ is

cyclic, the latter by **Euler's criterion** is equivalent to $n^{\frac{p-1}{2}} = 1 \in \mathbb{Z}_p^*$. Thus the primes p such that $n^{\frac{p-1}{2}} \neq 1 \in \mathbb{Z}_p^*$ can be discarded from the factor base in advance. Moreover, for the admissible odd primes p there are precisely two square roots $\pm\sqrt{n} \in \mathbb{Z}_p^*$ of $n \in \mathbb{Z}_p^*$, hence we have $f(x) = 0 \in \mathbb{Z}_p$ if and only if $x \in \{-m \pm \sqrt{n}\} \subseteq \mathbb{Z}_p$.

Now we apply a **sieve strategy** to $\mathcal{I}$, based on the following observation: If $p \in \mathbb{N}$ is a prime, then from $f(x+p) = (x+m+p)^2 - n = f(x) + p(2x+2m+p) \in \mathbb{Z}$ we conclude that $f(x) = 0 \in \mathbb{Z}_p$ if and only if $f(x+kp) = 0 \in \mathbb{Z}_p$ for all $k \in \mathbb{Z}$. Thus, we first compute $f(x) \in \mathbb{Z}$ for all $x \in \mathcal{I}$. Then, for the primes $p \in \mathcal{P}_{\leq b}$ in turn, we determine all $x \in \{-\frac{p-1}{2}, \ldots, \frac{p-1}{2}\}$ if p is odd, and $x \in \{0, 1\}$ for $p = 2$, such that $f(x) = 0 \in \mathbb{Z}_p$. For those we proceed through $\{x + kp \in \mathbb{Z}; k \in \mathbb{Z}\} \cap \mathcal{I}$ and divide out the maximum p-power dividing $f(x+kp) \in \mathbb{Z}$, and replace the latter by the quotient.

Assuming the validity of the Pomerance Conjecture, this yields a Las-Vegas algorithm to factor n, where we have **Pomerance's Theorem**: A proper divisor of n is found, using a factor base of size $l \sim \sqrt{L(n)}$ and needing $l^2 \sim L(n)$ tries, in expected sub-exponential time $O(L(n)^{1+o(1)})$.

For example, let $n := 7429$, hence we have $m := \lfloor\sqrt{n}\rfloor = 86$. We choose the factor base $[-1, 2, 3, 5, 7]$, that is $l := 4$; note that by Euler's Criterion all these primes are admissible. We choose the sieve interval $\mathcal{I} := \{-3, \ldots, 3\}$, that is $c := 3$; then the following shows that $[-3, 1, 2] \subseteq \mathcal{I}$ yields b-numbers:

x	-3	-2	-1	0	1	2	3
$(x+m)^2 - n$	-540	-373	-204	-33	140	315	492
sieve with 2	-135		-51		35		123
sieve with 3	-5		-17	-11		35	41
sieve with 5	-1				7	7	
sieve with 7					1	1	

The associated matrix of exponent vectors is $M := \begin{bmatrix} 1 & 2 & 3 & 1 & . \\ . & 2 & . & 1 & 1 \\ . & . & 2 & 1 & 1 \end{bmatrix} \in \mathbb{Z}^{3\times 5}$. Reduction yields $\begin{bmatrix} 1 & . & 1 & 1 & . \\ . & . & . & 1 & 1 \\ . & . & . & 1 & 1 \end{bmatrix} \in \mathbb{Z}_2^{3\times 5}$, whose kernel is $\langle [0, 1, 1] \rangle_{\mathbb{F}_2}$. This yields $x = (1+m) \cdot (2+m) = 87 \cdot 88 = 7656 \equiv 227 \pmod{n}$, and from $\frac{1}{2} \cdot [0, 1, 1] \cdot M = [0, 1, 1, 1, 1] \in \mathbb{Z}^5$ we get $y = (-1)^0 \cdot 2^1 \cdot 3^1 \cdot 5^1 \cdot 7^1 = 210$. Thus we obtain $\gcd(x-y, n) = 17$ and $\gcd(x+y, n) = 437$, where indeed $n = 17 \cdot 437$.

II Practical exercises (in German)

Protokollieren Sie Ihre Experimente und Ergebniss in einem Arbeitsheft, sowie in Text-Dateien, in die Sie auch Ihre '.sage'-Dateien kopieren.

7 Erste Schritte

(7.1) Aufgabe: Einloggen.
a) Um die folgenden Aufgaben bearbeiten zu können, müssen Sie sich zunächst auf einem der Rechner einloggen. Als Sitzungstyp können Sie etwa KDE wählen. Starten Sie eine Konsole, und einen Text-Editor, die Sie über das Start-Menü finden; als Editor können Sie etwa gedit oder XEmacs, allerdings keines der Office-Programme verwenden.

b) Die nachfolgenden Betriebssystemkommandos werden in der Konsole ausgeführt. Hilfen zu den Kommandos erhalten Sie mittels 'man ⟨command⟩'; mit den Pfeiltasten können Sie in der Anzeige scrollen, und sie mit 'q' verlassen. Probieren Sie die Hilfe zu den nachfolgenden Kommandos jeweils aus. Dabei werden Sie feststellen, daß die Kommandos jeweils (geschickt gewählte?) Abkürzungen sind. Fertigen Sie in Ihrem Arbeitsheft eine Liste der Kommandos, ihren Bedeutungen, und was sie jeweils bewirken an. Protokollieren Sie genau, welche Befehle Sie im folgenden ausführen.

(7.2) Aufgabe: Verzeichnisse und Dateien.
a) Orientieren Sie sich zunächst über die Verzeichnisstruktur Ihres Rechners; verwenden Sie dazu die Kommandos 'pwd', sowie 'cd' oder 'chdir'. Inhalte von Verzeichnissen können Sie sich mittels 'ls ⟨name⟩' ansehen; was macht 'ls'? Neue Verzeichnisse erzeugen Sie mittels 'mkdir ⟨name⟩'; erstellen Sie eines mit dem Namen 'blatt0'. Was bewirken die Kommandos 'cd blatt0' sowie 'cd ..' und 'cd'? Stellen Sie einen Teil der Verzeichnisstruktur (in Ihrem Arbeitsheft) graphisch dar, er sollte insbesondere Ihr eigenes Nutzerverzeichnis (auch 'Persönlicher Ordner' genannt) enthalten.

b) Benutzen Sie das Menü Ihres Editors, um eine Datei 'loeschen.txt' im Verzeichnis 'blatt0' zu öffnen, und die leere Datei zu speichern. Um sicher zu gehen, daß die Datei 'loeschen.txt' wirklich erzeugt wurde, sehen Sie sich den Inhalt des Verzeichnisses 'blatt0' an. Dateien löschen Sie mittels 'rm ⟨name⟩'. Löschen Sie die soeben erzeugte Datei wieder, und vergewissern Sie sich. Was passiert, wenn Sie so versuchen, das Verzeichnis 'blatt0' zu löschen?

c) Öffnen Sie nun eine Datei 'abgabe.txt' im Verzeichnis 'blatt0', kopieren Sie den bisherigen Inhalt der Konsole hinein, speichern Sie die Datei, und schließen Sie den Editor.

(7.3) Aufgabe: Prozesse.
a) Mittels 'ps' erhalten Sie Informationen über die Prozesse, die gerade auf

Ihrem Rechner laufen. Lassen Sie sich sowohl alle, als auch nur Ihre eigenen Prozesse anzeigen.

b) Starten Sie Ihren Editor erneut, und finden Sie seine ProcessId (PID) heraus. Prozesse können Sie mittels 'kill ⟨pid⟩' beenden. Damit können Sie etwa den Editor (unsauber) beenden. Was passiert, wenn Sie eine darin editierte Datei vorher nicht speichern?

(7.4) Aufgabe: SAGE.
a) Die nachfolgenden Aufgaben sollen mit dem Computeralgebra-System SAGE bearbeitet werden. Dazu starten Sie SAGE in einer Konsole mittels des Kommandos 'sage'. Warten Sie das Erscheinen des Prompts 'sage:' ab, dann können Sie mittels Eingaben in die Kommandozeile interaktiv mit SAGE rechnen. Mittels 'Ctrl-c' können Sie laufende Rechnungen, und mittels 'exit' oder 'Ctrl-d' können Sie SAGE ganz beenden.

b) Starten Sie die Browser-gestützte Einführung in SAGE mittels 'tutorial()'. Sehen Sie sich darin unter 'A Guided Tour' die Abschnitte 'Assignment, Equality, and Arithmetic', 'Getting Help' und 'Functions, Indentation, and Counting' an, und probieren Sie einige der dort genannten Beispiele aus. (Auch in Zukunft wird es hilfreich sein, wenn Sie sich weitere Abschnitte der Einführung und des Handbuchs 'manual()' ansehen.)

c) Im folgenden werden Sie häufig selbst SAGE-Programme schreiben. Um die grundlegenden Elemente der SAGE-Programmiersprache kennen zu lernen, sehen Sie sich in der Einführung unter 'Programming' die Abschnitte 'Loading and Attaching Sage files', 'Lists, Tuples, and Sequences' und 'Loops, Functions, Control Statements, and Comparisons' an, und probieren Sie einige der dort genannten Beispiele aus.

Die Benutzung von Systemen wie SAGE lernt man im übrigen am einfachsten spielerisch, durch Versuch und Irrtum, und es dauert nicht lange, bis es richtig Spaß macht! Falls es Ihnen so geht, und Sie mehr davon wollen, so können Sie SAGE (kostenlos und leicht) auf Ihrem eigenen Rechner installieren; besuchen Sie dazu etwa die Internet-Seite 'http://www.sagemath.org/de/'.

(7.5) Aufgabe: Fakultäten.
Für $n \in \mathbb{N}$ ist die **Fakultät** definiert als $n! := n \cdot (n-1) \cdots 2 \cdot 1$.

a) Berechnen Sie $n!$ für $n \in \{1, \ldots, 6\}$ interaktiv mit SAGE, und vergleichen Sie Ihre Ergebnisse mit denen der SAGE-Funktion factorial; zu dieser finden Sie Details im Handbuch.

b) Schreiben Sie Funktionen fak1, fak2 und fak3, die auf verschiedene Weisen $n!$ berechnen:
i) Verwenden Sie eine for-Schleife über einen geeigneten range.
ii) Verwenden Sie eine while-Schleife über einen geeigneten absteigenden Zähler.
iii) Verwenden Sie die Beziehung $n! = n \cdot (n-1)!$, für $n \geq 2$, für eine rekursive Funktion, die sich selbst aufruft.

Den Programmcode können Sie am besten in einer Text-Datei schreiben, die das Kürzel '.sage' trägt, und per 'load("⟨name⟩")' oder 'attach("⟨name⟩")' in SAGE eingelesen wird. Lesen Sie dazu nochmals die Kommentare in der Einführung zu den genannten Elementen der Programmiersprache, sowie zu Funktionsdefinitionen und Einrückungen; etwa sollte die erste Definition mit 'def fak1(n):' beginnen.

c) Vergleichen Sie für $n \in \{1, \ldots, 1000\}$ die Ergebnisse Ihrer drei Funktionen mit denen von factorial; hier bietet sich etwa eine Listenkonstruktion '[factorial(n) for n in range(1,1001)]' an.

(7.6) Aufgabe: Addition und Multiplikation.
Ermitteln Sie die Laufzeiten von Addition $x+y$ und Multiplikation $x \cdot y$ in SAGE, in Abhängigkeit von der Eingabegröße von $x, y \in \mathbb{N}$. Betrachten Sie etwa den Fall $x = y := 10^e$, wobei $e = k \cdot 10^l$ mit $k \in \{1, \ldots, 5\}$ und $l \in \{3, 4, 5\}$.

Verwenden Sie dazu die SAGE-Funktionen 'time' und 'cputime'. Um gute Zeitmessungen zu erhalten, führen Sie die jeweilige Operation in einer Schleife hinreichend oft aus. Interpretieren Sie die Ergebnisse. (Wieviel Zeit kostet etwa der Schleifendurchlauf?)

8 Teilbarkeit

(8.1) Aufgabe: Erweiterter Euklidischer Algorithmus.
a) Führen Sie den erweiterten Euklidischen Algorithmus (in Ihrem Arbeitsheft) für $a := 126$ und $b := 35$ aus; legen Sie dazu eine Tabelle an, in der sukzessive alle vorkommenden r_i, s_i, t_i, und q_i eingetragen werden.

b) Implementieren Sie den erweiterten Euklidischen Algorithmus als SAGE-Funktion (am besten also in einer '.sage'-Datei). Achten Sie dabei darauf, daß Sie jeweils nur die Variablen halten, die später noch verwendet werden. Außerdem können Sie die Anzahl der nötigen Schleifendurchläufe mittels eines geeigneten 'print'-Kommandos ausgeben. (Ihre Implementation werden Sie übrigens später weiter verwenden.)

Berechnen Sie mittels Ihrer SAGE-Funktion die größten gemeinsamen Teiler für einige der folgenden Beispiele; protokollieren Sie dazu auch die Anzahl der nötigen Schleifendurchläufe. Überprüfen Sie die gefundenen Bézout-Koeffizienten auf Korrektheit, und vergleichen sie Ihre Ergebnisse und Laufzeiten mit denen der SAGE-Funktion 'gcd'.

Berechnen Sie die obigen größten gemeinsamen Teiler auch mittels der naiven Methode, die die Faktorisierung der betrachteten Zahlen benutzt; verwenden Sie dazu die SAGE-Funktion 'factor'. Vergleichen Sie Ergebnisse und Laufzeiten mit Ihrer Implementation.

c) Die (aus allerlei Gründen mathematisch interessante) Folge $[F_n \in \mathbb{N}; n \in \mathbb{N}]$ der **Fibonacci-Zahlen** ist rekursiv definiert durch $F_1 := 1$ und $F_2 := 1$,

sowie $F_n := F_{n-1} + F_{n-2}$ für $n \geq 3$; in SAGE erhalten Sie diese Zahlen bereits vorgefertigt mittels der Funktion 'fibonacci'.

Berechnen Sie $\mathrm{ggT}(F_m, F_n)$ für einige $m, n \in \{1, \ldots, 10^9\}$. (Die Obergrenze ist kein Schreibfehler! Erst für große Zahlen wird der Euklidische Algorithmus richtig beeindruckend.)

d) Versuchen Sie, für die obigen Beispiele Gesetzmäßigkeiten zu entdecken, was die Anzahl der Schleifendurchläufe, sowie die gefundenen größten gemeinsamen Teiler angeht. Können Sie sie auch formal beweisen? (Computeralgebra hat in der Tat auch einen experimentellen Charakter: Durch Untersuchung vieler Beispiele kann man womöglich Muster erkennen...)

(8.2) Aufgabe: Binärer Euklidischer Algorithmus [Stein, 1967].
Es seien $a, b \in \mathbb{N}$. Dann gilt

$$\mathrm{ggT}(a,b) = \begin{cases} 2 \cdot \mathrm{ggT}(\frac{a}{2}, \frac{b}{2}), & \text{falls } a \text{ und } b \text{ gerade sind,} \\ \mathrm{ggT}(\frac{a}{2}, b), & \text{falls } a \text{ gerade und } b \text{ ungerade ist,} \\ \mathrm{ggT}(\frac{a-b}{2}, b), & \text{falls } a \text{ und } b \text{ ungerade sind.} \end{cases}$$

a) Beweisen Sie (in Ihrem Arbeitsheft) die Richtigkeit der obigen Relationen, entwickeln Sie daraus einen rekursiven Algorithmus zur Berechnung größter gemeinsamer Teiler, und führen Sie ihn für $a := 126$ und $b := 35$ aus.

b) Implementieren Sie Ihren Algorithmus als SAGE-Funktion, und vergleichen Sie seine Ergebnisse und Laufzeiten mit Ihrer Implementation des Euklidischen Algorithmus.

(8.3) Aufgabe: Sieb des Erathostenes.
a) Bestimmen Sie (in Ihrem Arbeitsheft) mit dem Sieb des Erathostenes alle Primzahlen ≤ 100.

b) Implementieren Sie das Sieb des Erathostenes als SAGE-Funktion, die für $n \in \mathbb{N}$ die Liste $[p \in \mathcal{P}; p \leq n]$ zurückgibt; versuchen Sie, eine möglichst effiziente Funktion zu schreiben.

c) Computeralgebra hat manchmal auch Wettbewerbscharakter. Hier ist eine Herausforderung: Wenn Sie glauben, daß Ihre Funktion schneller ist als die Ihrer Kommilitonen, so machen Sie das in Ihrer Abgabe kenntlich, und wir werden einen Vergleich machen; zugelassen sind nur Elemente der SAGE-Sprache, aber keine vorgefertigten Funktionen, und alle Rechnungen müssen im Bereich der ganzen Zahlen ablaufen. Dazu noch ein Hinweis: Ihre Funktion sollte für $n := 10^6$ in handhabbarer Zeit ein Ergebnis liefern.

d) Benutzen Sie Ihre Implementation des Sieb des Erathostenes, um die Funktion $\pi(n) := |\{p \in \mathcal{P}; p \leq n\}|$, für $n \in \mathbb{N}$, in SAGE zu implementieren; beachten Sie die SAGE-Funktion 'len'. Benutzen Sie die SAGE-Funktion 'plot_step_function',

um $\pi(n)$ jeweils für $n \in \{1, \ldots, 10^e\}$, wobei etwa $e \in \{1, \ldots, 4\}$, graphisch darzustellen. Was beobachten Sie? Versuchen Sie, experimentell eine glatte Funktion zu finden, die asymptotisch äquivalent zu $\pi(n)$ ist.

9 Modulare Arithmetik

(9.1) Aufgabe: Modulare Arithmetik.
a) Für $n \geq 2$ sei $\mathbb{Z}_n := \{0, \ldots, n-1\} \subseteq \mathbb{Z}$, und für $a \in \mathbb{Z}$ sei $\overline{a} \in \mathbb{Z}_n$ der Rest der Division von a durch n. Zeigen Sie (in Ihrem Arbeitsheft), die folgende Eigenschaft: Sind $a, a', b, b' \in \mathbb{Z}$ mit $\overline{a} = \overline{a'}$ und $\overline{b} = \overline{b'}$, so gilt auch $\overline{a+b} = \overline{a'+b'}$ und $\overline{ab} = \overline{a'b'}$. Folgern Sie daraus, daß die modulare Addition und Multiplikation die Assoziativ- und Distributivgesetze erfüllen.

b) Implementieren Sie SAGE-Funktionen, die Sie etwa add(a,b,n), mult(a,b,n) und inv(a,n) nennen, zur modularen Addition, Multiplikation, und zur Berechnung eines modularen Inversen (sofern eines existiert). Verwenden Sie zur Bestimmung modularer Inverser Ihre Implementation des erweiterten Euklidischen Algorithmus.

Bestimmen Sie als Anwendung alle $a \in \mathbb{N}_0$ mit $a < 1000$, so daß $67a$ in Dezimaldarstellung die drei letzten Ziffern 123 hat. Was passiert, wenn Sie stattdessen $68a$ und/oder 124 betrachten?

c) Es sei $\mathbb{Z}_n^*$ die Einheitengruppe von $\mathbb{Z}_n$. Zeigen Sie (in Ihrem Arbeitsheft), daß $1 \in \mathbb{Z}_n^*$ das einzige neutrale Element ist, und daß jedes Element von $\mathbb{Z}_n^*$ ein eindeutig bestimmtes inverses Element besitzt.

Untersuchen Sie die Werte $\varphi(n) = |\mathbb{Z}_n^*|$ der Eulerschen φ-Funktion für einige $n \in \mathbb{N}$, und versuchen Sie, Gesetzmäßigkeiten zu entdecken.

(9.2) Aufgabe: Exponentiation.
a) Führen Sie den 'Repeated-Squaring'-Algorithmus zur modularen Exponentiation (in Ihrem Arbeitsheft) für $n := 17$, sowie $a := 8$ und $e := 13$ aus.

b) Implementieren Sie eine SAGE-Funktion, etwa pow(a,e,n), zur Berechnung von $a^e \in \mathbb{Z}_n$, wobei $n \geq 2$, $a \in \mathbb{Z}_n$ und $e \in \mathbb{N}$, die 'Repeated Squaring' benutzt. Achten Sie dabei wiederum darauf, welche Variablen benötigt werden, und daß die Binärdarstellung des Exponenten $e \in \mathbb{N}$ nicht explizit berechnet wird.

Vergleichen Sie, für einige selbstgewählte (große) Beispiele, die Ergebnisse und Laufzeiten Ihrer Implementation mit dem naiven SAGE-Aufruf '$(a\hat{}e)\%n$'.

c) Für $n \in \mathbb{N}_0$ sei $F_n := 2^{2^n} + 1 \in \mathbb{N}$ die n-te **Fermat-Zahl**. Es fällt auf, daß $F_0 = 3$, $F_1 = 5$, $F_2 = 17$, $F_3 = 257$, $F_4 = 65537$ sämtlich Primzahlen sind, was [Fermat, 1640] veranlaßte, zu vermuten, daß alle Fermat-Zahlen prim sind. Aber nach [Euler, 1732] ist $F_5 = 4\,294\,967\,297$ nicht prim:

Zeigen Sie (in Ihrem Arbeitsheft), daß $641 \mid F_5$ gilt; verifizieren Sie das auch mittels Ihrer obigen SAGE-Funktion.

(9.3) Aufgabe: RSA-Cryptosystem mit Signatur.
a) Wählen Sie einen RSA-Modulus $n := pq < 10^{100}$, wobei $p \neq q$ Primzahlen sind; richten Sie es so ein, daß n nicht in handhabbarer Zeit mittels der SAGE-Funktion factor faktorisiert werden kann. (Wir hatten ja früher schon gesehen, daß es reichlich große Primzahlen gibt, diese sind also leicht zu finden; beachten Sie dazu die SAGE-Funktion next_prime.)

Wählen Sie weiter einen öffentlichen RSA-Schlüssel $e \in \mathbb{Z}_n$, und bestimmen Sie einen zugehörigen geheimen Schlüssel $d \in \mathbb{Z}_n$; richten Sie es so ein, daß d nicht leicht aus n und e allein berechnet werden kann.

b) Ziel ist es nun, eine verschlüsselte Nachricht zu senden, und diese gleichzeitig zu authentifizieren. Führen Sie dazu folgendes Signatur-Protokoll aus:

• Wählen Sie die Nachricht $a := m^i \in \mathbb{N}$, wobei $m \in \mathbb{N}$ Ihre Matrikelnummer und $i := \lfloor \log_m(n) \rfloor \in \mathbb{N}$ sind; ist diese Nachricht im gegebenen Kontext sinnvoll? (Eigentlich soll die gesendete Nachricht ja nicht öffentlich bekannt, sondern nur für den Empfänger zu entschlüsseln sein, aber hier dient das natürlich dazu, die Korrektheit Ihrer RSA-Implementation zu überprüfen.)

• Verschlüsseln Sie $a \in \mathbb{Z}_n$ zunächst mittels Ihres geheimen Schlüssels d, danach verschlüsseln Sie das Ergebnis $b \in \mathbb{Z}_n \subseteq \mathbb{Z}_N$ mit unserem unten genannten öffentlichen Schlüssel E. Senden Sie uns das Ergebnis $c \in \mathbb{Z}_N$ (in Ihrer E-Mail-Abgabe), zusammen mit Ihrem Modulus n und Ihrem öffentlichen Schlüssel e. (Wenn Sie gemeinschaftlich abgeben, so führen Sie das für alle Ihre Matrikelnummern aus; dabei brauchen Sie natürlich n und e nur einmal zu wählen.)

Hier unser öffentlicher Schlüssel: Wir wählen $E := 2^{16} + 1 = 65537$, und N ist die folgende 101-stellige Zahl:

$$16420157264974671169385955932174506230625625308062 6$$
$$96691855505073410692113405870505702894013562132361$$

c) Überlegen Sie sich (in Ihrem Arbeitsheft), wie aus den gesendeten Informationen die Nachricht zurückgewonnen werden kann, und wieso es sich gleichzeitig um eine Signatur handelt.

10 Primzahltests

(10.1) Aufgabe: Pseudo-Zufallszahlen.
Implementieren Sie eine SAGE-Funktion, die für $n \in \mathbb{N}$ sukzessive eine Folge $[x_1, x_2 \ldots]$ von **Pseudo-Zufallszahlen** in $\mathbb{Z}_n$ generiert, und bei jedem Aufruf das jeweils nächste Folgenglied zurückgibt. Gehen Sie dazu wie folgt vor:

Für Parameter $a \in \mathbb{Z}_n^*$ und $b \in \mathbb{Z}_n$, und einem Startwert $x_0 \in \mathbb{Z}_n$ sei $x_i := ax_{i-1} + b \in \mathbb{Z}_n$, für $i \in \mathbb{N}$, gegeben. Es bietet sich an, das jeweils aktuelle Folgenglied, mit dem SAGE-Kommando global, als globale Variable zu deklarieren.

Überlegen Sie sich zudem (in Ihrem Arbeitsheft), in welchem Sinne diese Folge zufällig ist.

(10.2) Aufgabe: Fermat-Test.
a) Für $n \in \mathbb{N}_0$ sei wieder $F_n := 2^{2^n} + 1 \in \mathbb{N}$ die n-te Fermat-Zahl; dabei sind bekanntlich $F_0, \ldots, F_4$ prim und F_5 zerlegbar, und man vermutet, daß F_n für $n \geq 5$ stets zerlegbar ist.

Zeigen Sie (in Ihrem Arbeitsheft), daß $3 \in \mathbb{Z}_{F_5}$ ein Fermat-Zeuge für die Zerlegbarkeit von F_5 ist. (Damit verifizieren wir die Zerlegbarkeit von F_5, finden aber die Faktorisierung nicht.)

b) Implementieren Sie eine SAGE-Funktion, die den probabilistischen Fermat-Test für $n \in \mathbb{N}$ und Fehlerwahrscheinlichkeit $0 < \epsilon < 1$ ausführt, und für zerlegbares n einen Fermat-Zeugen zurückgibt. Verwenden Sie dazu Ihre Implementation der modularen Exponentiation und Ihren Pseudo-Zufallszahlen-Generator.

c) Wenden Sie den Fermat-Test auf die Zahlen $n \in \{10 \cdot 10^6, \ldots, 11 \cdot 10^6\}$ an, jeweils mit Fehlerwahrscheinlichkeit $\epsilon := \frac{1}{2^k}$, für $k \in \{1, \ldots, 10\}$. Wieviele wahrscheinliche Primzahlen finden Sie jeweils? Wieviele davon sind Pseudo-Primzahlen (zu welchen Basen)? Wieviele davon sind Carmichael-Zahlen? (Zur Verifikation dürfen Sie hier die SAGE-Funktion is_prime oder Ihre Implementation des Lucas-Tests benutzen.)

d) Berechnen Sie Fermat-Zeugen für die Zerlegbarkeit von F_n, für $n \in \{5, 6, \ldots\}$. Für welche n ist das in handhabbarer Zeit machbar? Suchen Sie dabei nach möglichst 'kleinen' Fermat-Zeugen. (Wie müssen dazu die Parameter in Ihrem Pseudo-Zufallszahlen-Generator gewählt werden?) Was beobachten Sie?

(10.3) Aufgabe: Miller-Rabin-Test.
a) Implementieren Sie eine SAGE-Funktion, die den probabilistischen Miller-Rabin-Test für ungerades $n \in \mathbb{N}$ ausführt, und für zerlegbares n einen starken Zeugen zurückgibt. Verwenden Sie dazu Ihre Implementation der modularen Exponentiation und Ihren Pseudo-Zufallszahlen-Generator.

b) Wenden Sie den Miller-Rabin-Test auf die Zahlen $n \in \{10 \cdot 10^6, \ldots, 11 \cdot 10^6\}$ an, jeweils mit Fehlerwahrscheinlichkeit $\epsilon := \frac{1}{2^k}$, für $k \in \{1, \ldots, 10\}$, und vergleichen Sie die Ergebnisse und Laufzeiten mit Ihrer Implementation des Fermat-Tests. (Beachten Sie dabei die unterschiedliche Häufigkeit von Zeugen in den beiden Tests.)

Wieviele wahrscheinliche Primzahlen finden Sie jeweils? Wieviele davon sind starke Pseudoprimzahlen? Wieviele davon sind Carmichael-Zahlen? (Zur Verifikation dürfen Sie wieder die SAGE-Funktion is_prime oder Ihre Implementation des Lucas-Tests benutzen.)

c) Bestimmen Sie für $n \leq 11 \cdot 10^6$ alle starken Pseudoprimzahlen zur Basis 2 bzw. zu den Basen $\{2, 3\}$. Können Sie auch eine starke Pseudoprimzahl zu den Basen $\{2, 3, 5\}$ bzw. $\{2, 3, 5, 7\}$ finden?

Überlegen Sie sich (in Ihrem Arbeitsheft), wieso es sinnvoll ist, prime Basen zu betrachten.

(10.4) Aufgabe: Lucas-Test.
a) Bestimmen Sie (in Ihrem Arbeitsheft) Lucas-Zeugen für die Fermat-Zahlen F_n, wobei $n \in \{0, \ldots, 4\}$. (Hinweis: Suchen Sie wieder nach 'kleinen' Zeugen.)

b) Implementieren Sie eine SAGE-Funktion, die den den Lucas-Test für $n \in \mathbb{N}$ ausführt, und für unzerlegbares n ein Lucas-Zertifikat zurückgibt. Verwenden Sie dazu Ihre Implementation der modularen Exponentiation. Zur Faktorisierung von $n-1$ gehen Sie wie folgt vor:

Implementieren Sie eine SAGE-Funktion, die die 'kleinen' Primteiler von $n \in \mathbb{N}$ mittels Probedivision findet; dazu wiederum verwenden Sie Ihre Implementation des Sieb des Erathostenes, um (einmal) eine Liste der Primzahlen bis zu einer gegebenen Schranke, etwa 10^6, zu berechnen. Wir können also jetzt annehmen, daß n keine 'kleinen' Primteiler hat.

Verwenden Sie zun Ihre Implementation des Miller-Rabin-Tests, um n probabilistisch auf Unzerlegbarkeit zu prüfen. Falls danach n wahrscheinlich prim ist, so verwenden Sie den Lucas-Test rekursiv zur Verifikation. Falls n zerlegbar ist, so verwenden Sie die SAGE-Funktion factor; beachten Sie dabei, daß factor nur einen probabilistischen Primzahltest durchführt, so daß die so gefundenen 'Primteiler' auch mittels des Lucas-Tests rekursiv verifiziert werden müssen. (Wir lassen hier die SAGE-Funktion factor zu, weil wir noch keinen eigenen Faktorisierungsalgorithmus zur Verfügung haben.)

b) Ein **Pratt-Zertifikat [1975]** für die Unzerlegbarkeit von $n \in \mathbb{N}$ besteht aus einem Lucas-Zertifikat für n, sowie rekursiv aus Pratt-Zertifikaten für die Primteiler von $n-1$. (Für die gesicherten 'kleinen' Primzahlen in Ihrer Liste ist natürlich kein Zertifikat mehr erforderlich.)

Berechnen Sie Pratt-Zertifikate für die in Ihrem RSA-Modulus verwendeten Primzahlen. (Auch die SAGE-Funktion is_prime führt für große Zahlen nur einen probabilistischen Primzahltest durch, so daß Sie erst jetzt sicher sein können, daß die gewählten Zahlen prim sind.)

11 Faktorisierung

(11.1) Aufgabe: Die ρ-Methode.
a) Faktorisieren Sie (in Ihrem Arbeitsheft) $n := 1763$ mittels der ρ-Methode; verwenden Sie die Iteration $x \mapsto x^2 + 1$ und den Startwert $x := 1$, und führen Sie die Methode sowohl ohne als auch mit 'Cycle Detection' aus.

b) Implementieren Sie eine SAGE-Funktion, die für $n \in \mathbb{N}$ die ρ-Methode zur Berechnung eines nichttrivialen Teilers ausführt. Verwenden Sie dabei quadratische Funktionen zur Iteration, variieren Sie diese und den Startwert durch Parameter, und benutzen Sie 'Cycle Detection'.

Geben Sie auch die Anzahl der nötigen Schleifendurchläufe aus. Untersuchen Sie, anhand der folgenden Beispiele, wie diese Anzahl von der Wahl der Parameter abhängt, und wie sich diese Anzahl zur 'Quadratwurzel-Heuristik' verhält.

c) Für $n \in \mathbb{N}_0$ sei wieder $F_n := 2^{2^n} + 1 \in \mathbb{N}$ die n-te Fermat-Zahl; Sie ja bereits verifiziert, daß F_n für alle handhabbaren $n \geq 5$ zerlegbar ist. Berechnen Sie für $n \in \{5, \ldots, 12\}$ alle Primteiler $\leq 10^{14}$ von F_n. Versuchen Sie, Primteiler weiterer F_n zu finden, etwa für $n \leq 18$.

Wieviele Schleifendurchläufe sind nötig, um hinreichend sicher zu sein, alle Primteiler bis zu einer gegebenen Schranke gefunden zu haben? Verifizieren Sie (mittels Ihres Lucas-Tests), daß die gefundenen Teiler wirklich prim sind, und untersuchen Sie (mittels Ihres Miller-Rabin-Tests) die verbleibenden Kofaktoren von F_n auf Unzerlegbarkeit.

(11.2) Aufgabe: Die $(p-1)$-Methode.
a) Faktorisieren Sie (in Ihrem Arbeitsheft) $n := 1763$ mittels der $(p-1)$-Methode; verwenden Sie dabei die Glattheitsschranke $b := 5$. Was passiert, wenn Sie stattdessen $b := 7$ benutzen?

b) Implementieren Sie eine SAGE-Funktion, die für $n \in \mathbb{N}$ die $(p-1)$-Methode zur Berechnung eines nichttrivialen Teilers ausführt; behandeln Sie dabei die Glattheitsschranke als Parameter. Überlegen Sie sich (zunächst in Ihrem Arbeitsheft), wie die nötigen logarithmischen Schranken bestimmt und wie kleinste gemeinsame Vielfache berechnet werden können.

c) Versuchen Sie, die oben gefundenen Primteiler von F_n, für $n \in \{5, \ldots, 12\}$, wiederzuentdecken. (Wie können Ihnen die Ergebnisse der Lucas-Tests dabei helfen?) Untersuchen Sie, wie der Erfolg der Methode von der Wahl der Glattheitsschranke abhängt.

(11.3) Aufgabe: Die Dixon-Methode.
a) Für $n \in \mathbb{N}$ ungerade sei $f(X) := (X + \lfloor\sqrt{n}\rfloor)^2 - n$, und es sei $p \in \mathbb{N}$ prim mit $p \in \mathbb{Z}_n^*$. Zeigen Sie (in Ihrem Arbeitsheft): Es gilt $|\{x \in \mathbb{Z}_p; f(x) = 0 \in \mathbb{Z}_p\}| \in \{0, 1, 2\}$. Wann tritt welcher Fall ein?

b) Implementieren Sie eine SAGE-Funktion, die für ungerades $n \in \mathbb{N}$ die Dixon-Methode zur Berechnung eines nichttrivialen Teilers ausführt; verwenden Sie die Größe l der Faktorbasis und die Länge c des Suchintervalls als Parameter. Gehen Sie dabei wie folgt vor:

i) Zur Bestimmung einer Faktorbasis verwenden Sie Ihre Implementation des Sieb des Erathostenes, oder wahlweise die die SAGE-Funktion is_prime, um (einmal) eine Liste der Primzahlen bis zu einer geeigneten Schranke zu berechnen, und lassen Sie (abhängig von n) nur Primzahlen zu, die die Euler-Bedingung erfüllen.

ii) Sammeln Sie die ganzzahligen Exponentenvektoren in einer Liste von Listen auf, die dann als Matrix über $\mathbb{Z}_2$ aufgefaßt wird; beachten Sie dabei den Unterschied zwischen einer Liste von Listen und einer Matrix, und die SAGE-Funktion Matrix. Aus der Linearen Algebra ist bekannt, daß $\mathbb{Z}_2$ ein Körper ist, und wie man den Kern einer Matrix über einem Körper berechnet (Gauß-Algorithmus).

In SAGE werden endliche Körper mit dem Kommando 'GF(⟨size⟩)' erzeugt, mittels des Kommandos 'GF(⟨size⟩).coerce_map_from(ZZ)' kann man ganze Zahlen als Elemente eines endlichen Körpers auffassen, und zur Berechnung des Kerns einer Matrix dient die SAGE-Funktion kernel. (Die übernehmen wir, weil wir uns hier nicht mit algorithmischer linearer Algebra befassen wollen.) Sehen Sie sich dazu in der SAGE-Einführung unter 'A Guided Tour' auch die Abschnitte 'Linear Algebra' und 'Parents, Conversion and Coercion' an.

iii) Implementieren Sie zwei Varianten:

(P) Suchen Sie b-Zahlen systematisch in $\mathcal{I} := \{\lfloor\sqrt{n}\rfloor - c, \ldots, \lfloor\sqrt{n}\rfloor + c\}$, von der Mitte des Intervalls aus. (Hier dürfen Sie die SAGE-Funktionen sqrt und floor verwenden.) Dabei kann c variabel sein, und die Suche endet, sobald eine Relation gefunden wird; falls dies zu einer trivialen Faktorisierung führt, so kann etwa die zuletzt gefundene b-Zahl ignoriert und die Suche fortgesetzt werden.

(Q) Suchen Sie b-Zahlen mit dem eigentlichen Quadratischen Sieb, indem Sie für die Primzahlen p in der gewählten Faktorbasis auf einer Liste mit den Werten $f(x)$ in Schritten der Länge p entlanggehen, und nur für $x \in \mathcal{I}$ mit $f(x) = 0 \in \mathbb{Z}_p$ die p-Anteile aus $f(x)$ herausdividieren.

c) Untersuchen Sie, anhand der folgenden Beispiele, die Häufigkeit von b-Zahlen in Abhängigkeit von l und vom Abstand von der Mitte des Intervalls $\mathcal{I}$. Wie sollten die Parameter l und c in Abhängigkeit von n gewählt werden? Wie muß dabei c in Abhängigkeit von l variiert werden, und wie verändern sich damit die Laufzeiten? Wie verhalten sich gute Wahlen von l sich zur Dixon-Zahl $L(n)$? Vergleichen Sie die Laufzeiten der Varianten (P) und (Q), und vergleichen Sie das Verhalten der Dixon-Methode mit dem der ρ-Methode.

Hier sind nun einige Beispiele: (Weitere sind weiter unten zu finden.)

i) Für $n \in \mathbb{N}_0$ sei wieder $F_n := 2^{2^n} + 1 \in \mathbb{N}$ die n-te Fermat-Zahl. Faktorisieren Sie erneut F_n für $n \in \{5, 6, 7\}$, und versuchen Sie es für F_8. (Beachten Sie, wieviele Dezimalstellen diese Zahlen jeweils haben.)

ii) Faktorisieren Sie die folgenden Zahlen (vom RSA-Modulus-Typ):

$$\begin{aligned}
&4370161634111152737068 17\\
&2999541446900512353141818303\\
&4341853589553787843317594387 3373\\
&1992674160280932501870083141868 16507\\
&1483757509910600906323875397001481989077
\end{aligned}$$

(11.4) Aufgabe: Faktorisierung.
Nachdem nun auch Faktorisierungsalgorithmen zur Verfügung stehen, können Sie Ihre Implementation des Lucas-Tests zu einem eigenständigen Programmpaket aufbereiten, das in der Lage ist, Zahlen $n \in \mathbb{N}$ (mit Ausnahme der zerlegbaren Primzahlpotenzen) zu faktorisieren, und Pratt-Zertifikate für die Primteiler von n anzugeben.

Ihre Implementation sollte für beliebige Zahlen $n \leq 10^{35}$, und für weit größere Zahlen mit 'kleinen' Primteilern, handhabbare Laufzeit haben. (Ich finde es übrigens beeindruckend, daß man mit ein wenig raffinierter Mathematik auf den heute üblichen Rechnern so weit kommt.) Hier sind noch ein paar Beispiele:

Für $p \in \mathbb{N}$ sei $M_p := 2^p - 1 \in \mathbb{N}$ die p-te **Mersenne-Zahl**; für zerlegbares p ist M_p (offenbar) zerlegbar, aber man vermutet, daß es für p prim unendlich viele unzerlegbare Mersenne-Zahlen gibt. Untersuchen Sie die Zahlen M_p für Primzahlen $p \leq 200$: Welche davon sind (wahrscheinlich) prim? Faktorisieren Sie möglichst viele der anderen.