Application of Discrete Models

Adam Zlehovszky

December 21, 2023

1 Representation of Integers

1.1 Euclidean division

If $a, b \in \mathbb{Z}$ with $b \neq 0$ then $\exists! q, r \in \mathbb{Z}$ such that $a = qb + r$ where $0 \leq r < |b|$. This is the *Euclidean division* or *long division* of the *dividend* a with the *divisor* b. The results of the division are the *quotient* q and the *remainder* r. The standard notation for the remainder is $a \bmod b$. In algorithmic setting we use $q, r \leftarrow \mathbf{divmod}\,(a, b)$.

1.2 Number systems

Let $1 < b \in \mathbb{Z}$ be the *base* of the *number system.* For each $0 \leq n \in \mathbb{Z}$ there exists a unique $1 \leq d \in \mathbb{Z}$ and a unique set of *digits* $0 \leq n_1, n_2, \ldots, n_{d-1} < b$ all integers, such that

$$n = \sum_{k=0}^{d-1} n_k b^k.$$

If $n = 0$, then $d = 1$ and $n_0 = 0$. Otherwise $d = \lfloor \log_b n \rfloor + 1$ and we can extract the digits of n with long division, since

$$\begin{aligned} n &= n_{d-1} b^{d-1} + \cdots + n_2 b^2 + n_1 b + n_0 \\ &= \left(n_{d-1} b^{d-2} + \cdots + n_2 b + n_1\right) b + n_0 \end{aligned}$$

where the quotient $n_{d-1} b^{d-2} + \cdots + n_2 b + n_1$ is a $d-1$ digit number and n_0 is the extracted digit.

We call n_0 the *least significant digit* and n_{d-1} the *most significant digit.* The storage order of digits is called *little endian* if we start at the least significant digits and move towards the most significant one. Otherwise it is called *big endian.*

1.3 Operations on Integers

1.3.1 Addition

Let us assume that we have two unsigned integers stored as digits in a number system with base b:

$$n^{(i)} = \sum_{k=0}^{d^{(i)}-1} n_k^{(i)} b^k,$$

for $i = 1, 2$. The following algorithm computes the digits of the sum $s = n^{(1)} + n^{(2)} = \sum_{k=0}^{d^{(s)}-1} s_k b^k$:

Algorithm 1 Standard addition

procedure $\textsc{StandardAddition}(n^{(1)}, n^{(2)})$
  $d^{(s)} \leftarrow \max\left(d^{(1)}, d^{(2)}\right)$
  $c \leftarrow 0$
  for $k = 0, \ldots, d^{(s)} - 1$ **do**
    $c,\ s_k \leftarrow$ **divmod** $\left(n_k^{(1)} + n_k^{(2)} + c, b\right)$
  end for
  return s
end procedure

In Algorithm 1 we assume that $n_k^{(i)} = 0$ if $k \geq d^{(i)}$ for $i = 1, 2$. The time complexity of the standard addition is $O\left(d^{(s)}\right)$.

1.3.2 Multiplication

Let $n^{(i)}$'s defined same as above for $i = 1, 2$. We will compute the digits of the product $p = n^{(1)} \cdot n^{(2)} = \sum_{k=0}^{d^{(p)}-1} p_k b^k$ with the naive multiplication method:

The time-complexity of Algorithm 2 is $O(d^{(1)} \cdot d^{(2)}) = O(d^2)$, where $d = \max\left(d^{(1)}, d^{(2)}\right)$.

Karatsuba's idea for faster multiplication can be demonstrated on two-digit numbers. Let

$$x = x_1 b + x_0, \text{ and } y = y_1 b + y_0$$

with $0 \leq x_i, y_i < b$ integers. Naive multiplication of x and y is

$$\begin{aligned} z = xy &= (x_1 b + x_0)(y_1 b + y_0) \\ &= x_1 y_1 b^2 + (x_1 y_0 + x_0 y_1) b + x_0 y_0 \\ &= z_1 b^2 + z_1 b + z_0. \end{aligned}$$

This is 4 multiplication and 1 addition.

Algorithm 2 Naive multiplication

```
procedure NaiveMultiplication(n^(1), n^(2))
    d^(p) ← d^(1) + d^(2)
    for k = 0, ..., d^(p) − 1 do
        p_k ← 0
    end for
    for j = 0, ..., d^(2) − 1 do
        c ← 0
        for i = 0, ..., d^(1) − 1 do
            c, p_{i+j} ← divmod(p_{i+j} + n_i^(1) n_j^(2) + c, b)
        end for
        p_{d^(1)+j} ← c
    end for
    return p
end procedure
```

Now we can express

$$\begin{aligned} z_1 &= x_1 y_0 + x_0 y_1 \\ &= x_1 y_0 + x_0 y_1 - x_1 y_1 + x_1 y_1 - x_0 y_0 + x_0 y_0 \\ &= (x_1 + x_0)\, y_1 + (x_1 + x_0)\, y_0 - x_1 y_1 - x_0 y_0 \\ &= (x_1 + x_0)\,(y_1 + y_0) - x_1 y_1 - x_0 y_0 \\ &= (x_1 + x_0)\,(y_1 + y_0) - z_2 - z_0. \end{aligned}$$

This is 3 multiplication and 3 additions. By extending this idea to more than two digits recursively, the multiplication algorithm performs $O(d^{\log_2 3}) \approx O(d^{1.58})$ single-digit multiplication.

Fast Fourier Transform based algorithms can achieve $O(d \log d)$ complexity.

1.4 Exponentiation

We want to compute x^n for some $1 \leq n \in \mathbb{Z}$ and x that has multiplication as an operation.

Naive exponentiation By repeated multiplication, we can compute

$$x^n = \underbrace{x \cdot x \cdots x}_{n \text{ times}}.$$

This method requires $n - 1$ multiplications.

Repeated squaring If $n = 2^s$ for $0 < s \in \mathbb{Z}$, then

$$x^{(2^s)} = \left(x^2\right)^{\left(2^{s-1}\right)}.$$

This way we can compute x^n with $\log_2 n = s$ multiplications with the algorithm below:

Algorithm 3 Repeated squaring

```
procedure RepeatedSquaring(x, s)
    y ← x
    for k = 0, ..., s − 1 do
        y ← y^2
    end for
    return y
end procedure
```

Fast exponentiation If we write $n = \sum_{k=0}^{d-1} n_k 2^k$ in binary, then

$$\begin{aligned} x^n &= x^{\left(\sum_{k=0}^{d-1} n_k 2^k\right)} \\ &= \prod_{k=0}^{d-1} x^{\left(n_k 2^k\right)} \\ &= \prod_{k=0}^{d-1} x^{\left(2^k\right)^{n_k}}. \end{aligned}$$

Since $y^{n_k} = y$ if $n_k = 1$ and $y = 1$ otherwise, we arrive at the following algorithm:

Algorithm 4 Fast exponentiation

```
procedure FastExp(x, n)
    y ← 1
    while n > 0 do
        n, r ← divmod (n, 2)
        if r = 1 then
            y ← y · x
        end if
        x ← x^2
    end while
    return y
end procedure
```

This algorithm requires $O(\log_2 n)$ multiplication.

2 Number Theory and its Applications

2.1 Divisibility

If $aq = b$, then we say that a is a *divisor* of b, or b is a *multiple* of a. The notation is $a \mid b$. Otherwise $a \nmid b$. If $a \mid b$, then long division has remainder of 0 and in case of integers $\frac{b}{a}$ is an integer as well.

The following properties are natural consequences of the definition.

1. For every a, we have that $a \mid a$.
2. $a \mid 0$, for every a.
3. If $0 \mid a$, then $a = 0$.
4. If $a \mid b$ and $b \mid c$, then $a \mid c$.
5. If $a \mid b$ and $c \mid d$, then $ac \mid bd$.
6. If $a \mid b$, then $ac \mid bc$ for every c.
7. If $ac \mid bc$ and $c \neq 0$, then $a \mid b$.
8. If $a \mid b_i$ for some finite indices i, then $a \mid \sum_i c_i b_i$ for every c_i.

If $\varepsilon \mid a$ for every a, then we call ε a *unit element*. The unit elements of $\mathbb{Z}$ are ± 1.

If $a \mid b$ and $b \mid a$ and $a \neq b$, then we call a and b *associated elements*. Two elements a and b are associated if and only if $a = \varepsilon b$ for some unit element ε. Consequently, a and b are associated integers if and only if $|a| = |b|$.

Let $p \neq 0$ be a non-unit element. We say that p is an *irreducible element* if $p = ab$ implies that a or b is an associated element of p (and the other is a unit). If an element is not irreducible, then it is *composite*. We call p a *prime element* if $p \mid ab$ implies that $p \mid a$ or $p \mid b$. If p is an irreducible element, then it is also a prime element. In case of integers, the reverse is also true, i.e. every prime element is irreducible.

Theorem 1 (The Fundamental Theorem of Arithmetic)**.** *If $a \neq 0$ is not a unit element, then it is a product of irreducible elements. The product is unique (up to ordering and up to multiplication with unit elements).*

If $1 < n \in \mathbb{Z}$, then the *canonical form* of

$$n = p_1^{\alpha_1} \cdots \ p_r^{\alpha_r}$$

where p_i's are different prime numbers (positive prime elements of $\mathbb{Z}$) and $\alpha_i > 0$ for all $i = 1, \dots, r$.

Theorem 2 (Euclid)**.** *There are infinitely many prime numbers.*

Proof. Let us assume that there are only finite many primes $p_1, \ldots, p_n$. In this case the long division of $n = p_1 \cdots p_n + 1$ with p_i yields a remainder of 1 for every prime. This means that n does not have canonical form, which contradicts Theorem 1. $\square$

Theorem 3 (Distribution of prime numbers)**.** *The following statements illustrate some properties of the distribution of prime numbers:*

1. *If $N > 1$, then $\exists a > 2$ such that $a + 1, a + 2, \ldots, a + N$ are all composite numbers.*
2. *For every $M > 2$, there is a prime number between M and $2M$.*

Let $\pi(x)$ denote the number of positive prime numbers below x.

Theorem 4 (Prime Number Theorem)**.** *An approximation of $\pi(x)$ is $\frac{x}{\ln x}$. In other words*

$$\lim_{x \to +\infty} \frac{\pi(x)}{x / \ln x} = 1.$$

2.2 Greatest common divisor

The *greatest common divisor* of $a_1, \ldots, a_n$ elements is d, if

- $d \mid a_i$ for all i, i.e. d is a common divisor of the elements and
- if $d' \mid a_i$ for all i, then $d' \mid d$ that is d is maximal with respect to divisibility.

From the definition, it is clear that if d is a greatest common divisor and ε is a unit element, then εd is also a greatest common divisor. We usually fix a greatest common divisor for integers by nominating the positive one. We will use the notation $\gcd(a_1, \ldots, a_n)$.

From the definition we can see that

1. $\gcd(a, 0) = a$,
2. $\gcd(0, 0) = 0$,
3. $\gcd(a, b) = \gcd(b, a)$,
4. $\gcd(a, b) = \gcd(a - b, b)$ or in general $\gcd(a - qb, b) = \gcd(a, b)$ for any q. Specifically $\gcd(a \bmod b, b) = \gcd(a, b)$.

From the last property we have that

$$\begin{aligned}
\gcd(a, b) &= \gcd(a \bmod b, b) \\
&= \gcd(a \bmod b, b \bmod (a \bmod b)) \\
&= \gcd((a \bmod b) \bmod b \bmod (a \bmod b), b \bmod (a \bmod b)).
\end{aligned}$$

Let us define the recurrence relation $r_0 = a$, $r_1 = b$ and $r_{n+2} = r_n \bmod r_{n+1}$. In this case the equations above become

$$\begin{aligned} \gcd(r_0, r_1) &= \gcd(r_2, r_1) \\ &= \gcd(r_2, r_3) \\ &= \gcd(r_4, r_3). \end{aligned}$$

We can swap the arguments of gcd, so

$$\begin{aligned} \gcd(r_0, r_1) &= \gcd(r_1, r_2) \\ &= \gcd(r_2, r_3) \\ &= \gcd(r_3, r_4) \end{aligned}$$

The sequence in strictly decreasing, that is $r_n > r_{n+1}$ and $r_n \geq 0$ from $n = 2$. This means that there is an index N such that $r_N = 0$, but $r_{N-1} > 0$. From the properties and definition of recurrence relation it is clear, that

$$\gcd(a, b) = \gcd(r_{N-1}, 0) = r_{N-1}.$$

The argument above gives the *Euclidean algorithm* in recursive form:

Algorithm 5 Recursive Euclidean algorithm

procedure GCD(a, b)
    if $b = 0$ **then**
        return a
    end if
    return GCD(b, $a \bmod b$)
end procedure

We can transform the recursion into a loop. At any given step, we only need r_n and r_{n+1} to produce r_{n+2}.

Algorithm 6 Iterative Euclidean algorithm

procedure GCD(a, b)
    $r_{\text{old}},\ r_{\text{new}} \leftarrow a,\ b$
    while $r_{\text{new}} \neq 0$ **do**
        $r_{\text{old}},\ r_{\text{new}} \leftarrow r_{\text{new}},\ r_{\text{old}} \bmod r_{\text{new}}$
    end while
    return r_{old}
end procedure

For later applications we not only need the $\gcd(a, b) = d$ but two additional elements x and y, such that

$$d = ax + by.$$

We can extend Euclidean algorithm to calculate x and y. For this, we are going to ensure that during the algorithm we are updating producing x_n and y_n for each r_n, such that

$$r_n = ax_n + by_n.$$

This will be our loop invariant property, i.e. this property will be true for n and $n+1$ whenever we enter the body of the loop and we will make sure that it is true for $n+2$ after the last statement of the body. Before the first execution of the loop body, the values $x_0 = 1$, $y_0 = 0$, $x_1 = 0$ and $y_0 = 1$ will suffice. Let $q = \lfloor r_n / r_{n+1} \rfloor$. During the body of the loop we have that

$$\begin{aligned} r_{n+2} &= r_n \bmod r_{n+1} \\ &= r_n - q r_{n+1} \\ &= (ax_n + by_n) - q\,(ax_{n+1} + by_{n+1}) \\ &= a\,(x_n - qx_{n+1}) + b\,(y_n - qy_{n+1})\,, \end{aligned}$$

so $x_{n+2} = x_n - qx_{n+1}$ and $y_{n+2} = y_n - qy_{n+1}$ will work. Again, we only need tha last two value of x and y during the calculations. We call the procedure *Extended Euclidean algorithm*:

Algorithm 7 Extended Euclidean algorithm

procedure GCD(a, b)
    $r_{\text{old}},\ x_{\text{old}},\ y_{\text{old}} \leftarrow a,\ 1,\ 0$
    $r_{\text{new}},\ x_{\text{new}},\ y_{\text{new}} \leftarrow b,\ 0,\ 1$
    while $r_{\text{new}} \neq 0$ **do**
        $r_{\text{old}},\ q,\ r_{\text{new}} \leftarrow r_{\text{new}},\ \textbf{divmod}\,(r_{\text{old}}, r_{\text{new}})$
        $x_{\text{old}},\ x_{\text{new}} \leftarrow x_{\text{new}},\ x_{\text{old}} - qx_{\text{new}}$
        $y_{\text{old}},\ y_{\text{new}} \leftarrow y_{\text{new}},\ y_{\text{old}} - qy_{\text{new}}$
    end while
    return $r_{\text{old}},\ x_{\text{old}},\ y_{\text{old}}$
end procedure

2.3 Linear Diophantine equation

Let $a, b, c \in \mathbb{Z}$ and we search for the integers solutions $x, y \in Z$ for the equation

$$ax + by = c.$$

If $d = \gcd(a, b)$ then $d \mid ax + by$ there $d \mid c$. Otherwise there is no solution. With Extended Euclidean algorithm, we can compute $d = ax' + by'$. Since $d \mid c$, there exists $m \in \mathbb{Z}$ for which $c = d \cdot m$. In summary

$$a\,(x'm) + (x'm) = d \cdot m = c,$$

so $x_0 = x'm$ and $y_0 = y'm$ is a pair of solution.

If there is a pair of solution, then there are infinitely many solutions in the form of

$$x = x_0 + k\frac{b}{d}, \quad y = y_0 - k\frac{a}{d}$$

where $k \in Z$ and there are no other solutions.

2.4 Modular arithmetic

Let $m > 1$. We say that a is *congruent to* $b \pmod m$, if $m \mid a-b$. The notation for this relation $a \equiv b \pmod m$. Otherwise we say, that a is *incongruent to* b, or $a \not\equiv b \pmod m$.

For a fixed $m > 1$, $a \equiv b \pmod m$ defines an equivalence relation over the $\mathbb{Z}$. The equivalence classes are called *residue class*es. The residue class of a is

$$\overline{a} = \{a + km : k \in \mathbb{Z}\}.$$

Let us denote

$$\mathbb{Z}_m = \left\{\overline{0}, \overline{1}, \ldots, \overline{m-1}\right\}$$

the set of residue classes.

Theorem 5 (The compability of integer operations with the residue classes)**.** *If* $a \equiv b \pmod m$ *and* $c \equiv d \pmod m$*, then*

- $a + c \equiv b + d \pmod m$ *and*
- $ac \equiv bd \pmod m$.

This can be stated on residue class level as well:

1. $\overline{a} + \overline{b} = \left\{a' + b' : a' \in \overline{a}, b' \in \overline{b}\right\} = \overline{a+b}$ *and*
2. $\overline{a} \cdot \overline{b} = \left\{a'b' : a' \in \overline{a}, b' \in \overline{b}\right\} = \overline{ab}$.

Theorem 5 enables us to work on the *residue system* level, i.e. on

$$\mathbb{Z}_m = \left\{\overline{0}, \overline{1}, \ldots, \overline{m-1}\right\} \simeq \{0, 1, \ldots, m-1\}$$

The operations on the residue system can be defined as

$$a +_m b = (a+b) \bmod m, \quad a \cdot_m b = ab \bmod m.$$

Theorem 6 (Structure of $\mathbb{Z}_m$)**.** *Let* $m > 1$ *and* $a \neq 0$.

1. *If* $\gcd(a, m) \neq 1$*, then* $ax \equiv 0 \pmod m$ *has non-zero solution.*
2. *If* $\gcd(a, m) = 1$*, then* $ax \equiv 1 \pmod m$ *has a solution.*

Proof. 1. If $d = \gcd(a, m) > 1$, then

$$\frac{a}{d}m = a\frac{m}{d} \equiv 0 \pmod m,$$

so $x = m/d$ is nonzero solution.

2. With the extended Euclidean algorithm, we can compute $1 = ax + my$. Rearranging this, we have that $ax - 1 = -ym$. So $m \mid ax - 1$, which is by definition means that $ax \equiv 1 \pmod{m}$.

$\square$

If $ax \equiv 1 \pmod{m}$, then we call x the *modular inverse* of a, denoted by $a^{-1} \pmod{m}$.

2.5 Solving linear congruence equations

We search for the integer solutions of $ax \equiv b \pmod{m}$. We can convert this expression into the linear Diophantine equation $ax + my = b$. We know that $d = \gcd(a, m) \mid b$, that is $b = s \cdot d$ otherwise there is no solution.

A particular x_0 can be found by extended Euclidean algorithm. If $d = ax' + my'$, then $x_0 = x's$ and the general solutions have the form of $x = x_0 + k\frac{m}{d}$ for integers k.

We know that $x \equiv x + m \pmod{m}$ and for every $1 < n < m$, $x \not\equiv x + n \pmod{m}$. This means if $1 < k\frac{m}{d} < m$, then $x \not\equiv x + k\frac{m}{d} \pmod{m}$. So we can list all the different solutions $\pmod{m}$ with $k = 0, \ldots, d - 1$.

2.6 Solving systems of linear congruence relations

No we are searching for common integer solution of the following two linear congruence equations

$$z \equiv a \pmod{c}, \quad z \equiv b \pmod{d},$$

where $a, b \in \mathbb{Z}$ and $0 < c, d \in Z$.

Theorem 7 (Chinese Remainder Theorem (CRT))**.** *If* $\gcd(c, d) = 1$*, then the system above has a solution and the solution is unique* $\pmod{cd}$*.*

Proof. We only prove the existence.

With extended Euclidean algorithm we can compute $cx + dy = 1$. If we let $z = bcx + ady$, then

$$bcx + ady \equiv 0 + ady \equiv ady \equiv a \pmod{c},$$

since $c \mid bcx$ and $y = d^{-1} \pmod{c}$. The other congruence relation can proven in the same way. $\square$

We can extend this method for solving more than two equations by replacing two of them with the common solution calculated with the help of previous theorem.

2.7 Euler's totient function

For $0 < n \in \mathbb{Z}$ we can define *Euler's totient function* as

$$\varphi(n) = |\{a \in \mathbb{Z} : 1 \leq a \leq n, \gcd(a,n) = 1\}|.$$

If p is a positive prime number, then $\varphi(p) = p - 1$.

If $1 < k \in Z$, then $\gcd(a, p^k) > 1$ means that $p \mid a$ for integers $1 \leq a \leq p^k$. This means that $a = p, 2p, \ldots, p^{k-1}p$. Due to this reasoning, we have that

$$\varphi\left(p^k\right) = p^k - p^{k-1} = p^k\left(1 - \frac{1}{p}\right).$$

If $\gcd(n, m) = 1$, then $\varphi(nm) = \varphi(n)\varphi(m)$. This last statement is the consequence of CRT.

In summary, if the canonical form of $1 < n = p_1^{\alpha_1} \cdots p_r^{\alpha_r}$, then

$$\varphi(n) = n \prod_1^r \left(1 - \frac{1}{p_i}\right).$$

Theorem 8 (Euler-Fermat)**.** *If $1 < n \in Z$ and $gcd(a, n) = 1$ for $a \in Z$, then $a^{\varphi(n)} \equiv 1 \pmod{n}$.*

Theorem 9 (Fermat's little theorem)**.** *If $1 < p \in Z$ is a prime number and $a \in Z$, then $a^p \equiv a \pmod{p}$. If in addition $p \not| a$, then $a^{p-1} \equiv 1 \pmod{p}$.*

2.8 Probabilistic primality testing

If p is a prime number, then for any $1 < a < n-1$, due to Euler-Fermat Theorem $a^{p-1} \equiv 1 \pmod{p}$. This statement is known as *Fermat's little theorem* and we can use it as primality testing for $n > 2$ odd integer:

1. Select randomly $a \in_R \{2, \ldots, n-2\}$.
2. If $\gcd(a, n) = 1$ and $a^{n-1} \equiv 1 \pmod{n}$, then n is probably a prime number. Otherwise it is composite.

There are examples of such composite numbers n that for every base a, where $\gcd(a, n) = 1$ the property $a^{n-1} \equiv 1 \pmod{n}$ still holds. These are called *Carmichael numbers* and there are infinitely many of them.

We can use additional properties for further testing.

Theorem 10. *A positive integer p is a prime number if and only if the only solutions of $x^2 \equiv \pmod{p}$ are $x \equiv \pm 1 \pmod{m}$. On the residue system level this means that p is a prime number if and only if in $\mathbb{Z}_p$ the square roots of 1 are 1 and $p - 1$.*

Let $n-1 = 2^s q$, where q is odd. Combining Fermat's little theorem and the previous property of prime numbers, the sequence

$$a^{2^s q} \mod n, a^{2^{s-1} q} \mod n, \ldots, a^q \mod n$$

must start with at least one 1 and if there are any elements not equal to 1, the first such element must be $n-1$.

Based on these arguments we arrive the Miller-Rabin primality testing algorithm. The probability of false positive response $p < \frac{1}{4}$. If we repeat the trial k times independently, then $p < \frac{1}{4^k}$.

Algorithm 8 Miller-Rabin probabilistic primality test

procedure MILLERRABIN(n)
    Compute integers s, q such that $n-1 = 2^s q$ and q is odd.
    Select $a \in_R \{2, \ldots, n-2\}$.
    $x \leftarrow a^q \mod n$ ▷ Use fast modular exponentiation
    if $x = 1$ **then**
        return probably prime
    end if
    for $k = 0, \ldots, s-1$ **do**
        if $x = n-1$ **then**
            return probably prime
        end if
        $x \leftarrow x^2 \mod n$.
    end for
    return composite
end procedure

2.9 Discrete logarithm problem

Let us fix a prime number p. We will denote the nonzero elements of $\mathbb{Z}_p$ with $\mathbb{Z}_p^*$. The *(multiplicative) order* of $a \in \mathbb{Z}_p^*$, denoted by $\mathrm{ord}_p(a)$ is defined by

$$\mathrm{ord}_p(a) = \min \{0 < n \in \mathbb{Z} : a^n \mod p = 1\}.$$

Due to Fermat's little theorem ($a^{p-1} \mod p = 1$), the order of a always exists and $\mathrm{ord}_p(a) \leq p-1$. If $a^i \mod p = a^j \mod p$ with $i > j$, then a^{i-j} mod 1. This means $\mathrm{ord}_p(a) \leq i-j$. From this, we can see that if $\mathrm{ord}_p(a) = n$, then the elements $a, a^2, \ldots, a^n$ are all different. If the order the element g is maximal, i.e. $\mathrm{ord}_p(g) = p-1$, then $\mathbb{Z}_p^* = \left\{g, g^2, \ldots, g^{p-1}\right\}$. In this case we call g a *generator* element of Z_p^*.

Let g be a generator element of $\mathbb{Z}_p^*$, $k \mid p-1$ with $0 < k < p-1$ and $p-1 = kn$ for some n integer. In this case $g^{p-1} \mod p = (g^k)^n \mod p = 1$ and $g^k \mod p \neq 1$. If $s \mid k$, then $s \mid p-1$, so $g^s \mod p \neq 1$ as well.

This reasoning gives us the following way of testing whether an element g is generator or not: If k is a maximal proper divisor of $p-1$ (k does not divide any other factor of $p-1$), then $g^k \mod p \neq 1$. If the canonical form of $p-1 = p_1^{\alpha_1} \cdots p_r^{\alpha_r}$, then the maximal divisors of $p-1$ are

$$\frac{p-1}{p_i} \qquad i = 1, \ldots, r.$$

We call p a *Sophie Germain prime*, if it is a prime number and $p = 2q+1$, where q is also a prime number. In this case the it is enough to test $g^2 \mod p \neq 1$ and $g^q \mod p \neq 1$.

Let $b \in \mathbb{Z}_p^*$, $0 \leq k \leq p-1$ and $a = b^k \mod p$. In this case the *discrete logarithm* or *index* of a in base b in $\mathbb{Z}_p^*$ is k. The notation is $\log_b a = k$ or $\mathrm{ind}_b a = k \pmod p$.

The discrete logarithm problem (DLP) is to find k from b and a. Currently there is no efficient algorithm known to compute k.

2.9.1 Diffie-Hellman key exchange

Let us fix a large Sophie Germain prime number p and search for a generator element g. Alice secretly selects $1 \leq a \leq p-1$ and Bob does the same with $1 \leq b \leq p-1$. Alice computes $A = g^a \mod p$ and Bob computes $B = g^b \pmod p$. They share A and B between each other. Alice computes $B^a \pmod p$, Bob computes $A^b \pmod p$. Since

$$B^a = (g^b)^a = g^{ba} = g^{ab} = (g^a)^b = A^b,$$

they have common secret key. The security of the key exchange depends on that we assume DLP cannot be solved efficiently.

3 RSA

Let us fix two large prime numbers, p and q and calculate their product $n = pq$. Also search for such integers e and d such that $ed \equiv 1 \mod \varphi(n)$, i.e. $ed = k(p-1)(q-1)+1$ for some $k \in Z$. We can make n and e public and keep p, q and d private. If $0 < m < n$ and $c = m^e \mod n$, then

$$c^d = (m^e)^d = m^{ed} = m^{k(p-1)(q-1)+1} = (m^{p-1})^{k(q-1)} m \equiv m \pmod p,$$

since $m^{p-1} \equiv 1 \pmod p$. By the same argument we have that $m \equiv c^d \pmod q$. We can solve this system with the Chinese remainder theorem and the solution is $m = c^d \mod n$.

Let $d_p = d \mod (p-1)$ and $d = k(p-1) + d_p$. In this case

$$c^d = c^{k(p-1)+d_p} = (c^{p-1})^k c^{d_p} \equiv c^{d_p} \pmod p.$$

Also if $d_q = d \mod (q-1)$, then $c^d \equiv c^{d_q} \pmod q$. Let $m_1 = c^{d_p} \mod p$ and $m_2 = c^{d_q} \pmod q$. In this case if m solves the system

$$m \equiv m_1 \pmod p, \qquad m \equiv m_2 \pmod q,$$

then m solves the original problem.

Let us search for the solution tin the form of $m = m_2 + hq$. In this case $m \equiv m_2 \pmod q$. If $m_2 + hq \equiv m_1 \pmod m$, then $hq \equiv m_1 - m_2 \pmod p$. Let us denote $q_{\text{inv}} = q^{-1} \pmod p$. In this case with $h = q_{\text{inv}}(m_1 - m_2) \mod p$, then m solves the system.

The security of RSA stands on the assumption that the factorization of n into p and q cannot be done efficiently. Due to this fact calculating $\varphi(n)$ without knowing p and q is hard as well. This means that even by knowing e, it is hard to compute d.

We can use RSA to sign messages digitally. If $\sigma = m^d \pmod n$, then we can check the signature of the the m, σ pair by checking that $m \equiv \sigma^e \pmod n$.

4 Polynomials over Fields

Our main requirement from a field F is that the linear equation $ax = b$ with $a, b \in F$, $a \neq 0$ has a solution $x \in F$. We calculate the solution by cancelling a, i.e. $x = a^{-1}b$, where a^{-1} is the multiplicative inverse of a.

The main examples of fields are $\mathbb{C}, \mathbb{R}, \mathbb{Q}$ and $\mathbb{Z}_p$ for prime p.

Informally, we can think of a polynomial as an expression in the form of

$$f = f_0 + f_1 x + f_2 x^2 + \cdots + f_n x^n.$$

Here f is the *polynomial*, f_i is a *coefficient*, $f_i x^i$ is a *term*, f_0 term is the *constant term* and f_n is the *main coefficient*. We call x the indeterminate variable. If the coefficients are all from the field F, then f is a polynomial over the field F. The set of all polynomials over F is denoted by $F[x]$. Sometimes we consider the polynomials as infinite sums of terms, where the coefficients are zero after $f_n \neq 0$, i.e.

$$f = f_0 + f_1 x + f_2 x^2 + \cdots + f_n x^n + 0x^{n+1} + 0x^{n+1} + \cdots$$

Since we can map every $F \ni c \mapsto c + 0x + 0x^s + \cdots$, every constant is a polynomial also. We can write $x = 0 + 1x + 0x^2 + \cdots$, therefore $x \in F[x]$.

The *degree* of f is defined by

$$\deg(f) = \begin{cases} -1, & f = 0 \\ 0, & f = c \in F, c \neq 0 \\ n, & f_n \neq 0, f_{n+k} = 0, 0 < k \in \mathbb{Z}. \end{cases}$$

In some sense, we can say that if the degree of polynomial is greater than the degree of another, then the first one is "larger".

4.1 Operations of $F[x]$

We work with polynomials as expression, we can add and multiply them. Since we can identify a polynomial with its list of coefficients, we must determine the coefficients of the result of addition and multiplication from the coefficients of the operands.

Let

$$f = f_0 + f_1x + \cdots + f_nx^n,$$

and

$$g = g_0 + g_1x + \cdots + g_mx^m + 0x^{m+1} + \cdots 0x^n.$$

Addition After collecting the term h_kx^k with $h = f+k$, we have that $h_kx^k = f_kx^k + g_kx^k$, so $h_k = f_k + g_k$. For the degree of $f+g$ we have that $\deg(f+g) \leq \max\{\deg(f), \deg(g)\}$.

Multiplication Again, we collect the term h_kx^k with $h = fg$. We have that

$$h_kx^k = f_0g_kx^k + f_1xg_{k-1}x^{k-1} + \cdots + f_kx^kg_0,$$

so

$$h_k = \sum_{i=0}^{k} f_ig_{k-i} = \sum_{j=0}^{k} f_{k-j}g_j = \sum_{i+j=k} f_jg_j.$$

The degree of fg is $\deg(f) + \deg(g)$.

Theorem 11 (Euclidean divison in of polynomials over fields)**.** *If $f, g \in F[x]$ with $g \neq 0$, then there exists a unique pair $q, r \in F[x]$ such that $f = qg + r$ with* $\deg(r) < \deg(g)$.

Proof. We only prove the existence. Let $n = \deg(f)$ and $m = \deg(g)$. If $n < m$, then $q = 0$, $r = f$ works.

Otherwise for the polynomial $f^* = f - f_ng_m^{-1}x^{n-m}g$ we have that $\deg(f^*) < \deg(f)$. We repeat this process until $r = \deg(f^*) < m$ and record $f_ng_m^{-1}x^{n-m}$ into q. □

4.2 Error detection with polynomials over $\mathbb{Z}_2$

We want to send the bits $b_0b_1 \ldots b_{k-1}$ over a noisy channel. We assume that we receive the same number of bits, but the values might change. On the receiver's end we want to be able to detect any error in communication. For this reason we introduce redundant information $r_0r_1 \ldots r_{m-1}$ into our bits.

The simples case is introducing only one bit. Let

$$r_0 = \sum_{i=0}^{k-1} b_i \mod 2.$$

If we send $r_0b_1 \ldots b_{k-1}$, then we always send even number of bits that are set to 1. We reject the received message if there are odd number of bits to to 1 in it. This method is called *parity check*, r_0 is the *parity bit*. This simplest form of error detection method is able to detect all such errors where odd number of bits are flipped. But it is incapable to detect even number of errors.

We can think of the bits $b_0b_1\dots b_{k-1}$ as the coefficients of the polynomial

$$b = b_0 + b_1x + \cdots b_{k-1}x^{k-1} \in \mathbb{Z}_2[x].$$

If we denote $r_0r_1\dots r_{m-1}b_0\dots b_{k-1}$ with $r||b$, then

$$r||b = r_0 + r_1x + \cdots r_{m-1}x^{m-1}b_0x^m + \cdots b_{k-1}x^{k-1+m} = r + bx^m.$$

If $g \in \mathbb{Z}_2[x]$ with $\deg(g) = m$, then we can write $bx^m = qg + r$ where $\deg(r) < m$. From this we, have $bx^m - r = qg$, so $g \mid bx^m - r$. But in $\mathbb{Z}_2[x]$, the polynomials $bx^m - r$ and $bx^m + r$ are the same. If we send the coefficients of $bx^m + r$, then on the receiver's end we can check if $g \mid bx^m + r$ still stands.

This error detection method is called *Cyclic redundancy check* or *CRR* and g is a *CRC polynomial*.

4.2.1 Designing CRC polynomials

We can think of the error of the transmission e as a polynomial over $\mathbb{Z}_2$. So we receive $bx^m + r + e$. To be able to detect the error, we must design g such that $g \nmid e$. Some things to consider:

- The degree of g influences the error detection capabilities, for e.g. the length of messages that can be sent in one go.
- In case of one bit error, we have that $e = x^s$. If g has at least two bits set, i.e. $x^i + x^j \mid g$, then $g \nmid x^s$.
- Two bit errors can written as $x^i + x^j = x^i(1 + x^j)$. We have already covered the x^i term. To protect against $(1 + x^j)$ we should make $1 + x^k \mid g$ with as big k as possible.
- The CRC polynomial $x + 1$ is the parity check method. If $x + 1 \mid g$, then CRC detects all odd number of errors.
- All burst error of n bits is detected a polynomial with $g_0 = 1$ and $\deg(g) \geq n$.

4.3 Number Theory of $F[x]$

$F[x]$ behaves just like $\mathbb{Z}$ so we can define divisibility. The units of $F[x]$ are the nonzero constant polynomials, since $F[x] \ni f = cg$ with $g_k = f_kc^{-1}$. The associated elements of f are cf for $0 \neq c \in F$. The irreducible and prime elements of $F[x]$ are the same. The fundamental theorem of arithmetic can be stated for polynomials as well. We can write every polynomial of $F[x]$ as uniquely as product of irreducible polynomials, i.e.

$$f = f_n \left(g^{(1)}\right)^{\alpha_1} \cdots \left(g^{(r)}\right)^{\alpha_r},$$

where $g^{(i)} \in F[x]$ is irreducible, with main coefficient of 1, $0 < \alpha_i \in \mathbb{Z}$ for $i = 1, \ldots, r$.

Polynomials have greatest common divisor, we can distinguish one by taking the one with main coefficient of 1. Extended Euclidean algorithms works for polynomials.

We can define congruence relation with a fixed modulus non-constant $m \in F[x]$. We can distinguish a unique element from every residue class with degree less than $\deg(m)$. These elements form the residue system $F[x]_m$.

The structure of $F[x]_m$ is the same as with $\mathbb{Z}_m$. If $\gcd(a, n) = 1$, then a has modular inverse which can be computed with extended Euclidean algorithm.

If m is irreducible, then every non-zero element in $F[x]_m$ has a modular inverse, so $F[x]_m$ is a field.

4.4 Roots of Polynomials

If $f \in F[x]$ and $r \in F$ then we can define the mapping $r \mapsto f(r) = \sum_{i=0}^{n} f_i r^i$. This is the *polynomial function* defined by the polynomial f. We denote the function by $\hat{f}$.

Note that $x, x^p \in \mathbb{Z}_p[x]$ are different polynomials, but $r^p \equiv r \pmod{p}$ so they share the same polynomial function.

If for $c \in F$ we have that $f(c) = 0$, then we call c the *root* of f.

Theorem 12. *If $0 \neq f \in F[x]$, $c \in F$ and $f(c) = 0$, then $x - c \mid f$.*

Proof. With Euclidean division we can write $f = (x - c)q + r$. Since $\deg(r) < \deg(x - c) = 1$, we have that $r = c' \in F$. But $0 = f(c) = (c - c)q + c'$, so $c' = 0$. □

This theorem has the following consequences:

1. If $0 \neq f \in F[x]$, then f has at most $\deg(f)$ roots.
2. If the polynomial functions of $f, g \in F[x]$ with $\deg(f), \deg(g) \leq n$ agree on $n + 1$ points, then $f = g$. Otherwise $f - g$ would have $n + 1$ roots.
3. If F is infinite, then two polynomials cannot share a polynomial function. Otherwise the difference polynomial would have infinite roots.

If $0 \neq f \in F[x]$, $1 \leq n \in \mathbb{Z}$. The c is the root of f *multiplicity* of n if $(x - c)^n \mid f$, but $(x - c)^{n+1} \nmid f$.

Theorem 13 (Lagrange interpolation)**.** *If $c_0, c_1, \ldots, c_n \in F$ are different and $d_0, d_1, \ldots, d_n \in F$, then there is a unique $f \in F[x]$ such that $f(c_j) = d_j$ for $j = 0, \ldots, n$. The j-th Lagrange base polynomial of the points $c_0, c_1, \ldots, c_n$ is defined by*

$$l_j = \frac{\prod_{i \neq j}(x - c_j)}{\prod_{i \neq j}(c_i - c_j)},$$

and $f = \sum_{j=0}^{n} d_j l_j$.

4.5 Irreducible polynomials

Theorem 14 (The Fundamental Theorem of Algebra)**.** *If* $1 \leq n \in \mathbb{Z}$*,* $f \in \mathbb{C}[x]$ *with* $\deg(f) = n$*, then there exists* $c \in \mathbb{C}$ *such that* $f(c) = 0$*.*

The irreducible polynomials of $\mathbb{C}[x]$ exactly the linear polynomials.

The irreducible polynomials of $\mathbb{R}[x]$ are the linear polynomials and such quadratic polynomials that do not have real root, for. e.g. $x^2 + 1$.

If p is a prime number and $1 \leq n \in \mathbb{Z}$, then $x^n - p \in \mathbb{Q}[x]$ is irreducible.

For every $1 \leq n \in \mathbb{Z}$, there exists $f \in \mathbb{Z}_p[x]$ such that f is irreducible.

4.6 Field extensions

Given and irreducible polynomial $m \in F[x]$, $F[x]_m$ is a field.

Let $F = \mathbb{R}$ and $m = x^2 + 1$. In this case the elements of the residue system are $ax + b$ for $a, b \in \mathbb{R}$. Now the product of two elements $\pmod{x^2+1}$ is

$$\begin{aligned} (ax+b)(cx+d) \mod x^2+1 = &= ac + (bc+ad)x + bdx^2 \mod x^2+1 \\ &= ac + (bc+ad)x + bdx^2 - bdx^2 - bd \\ &= (ac - bd) + (bc+ad)x. \end{aligned}$$

This means that $\mathbb{R}[x]_{x^2+1} \cong \mathbb{C}$.

For $\mathbb{Z}_p[x]_m$ with irreducible m with degree of n, the residue systems consists of the elements

$$f = f_0 + f_1x + f_2x^2 + \cdots + f_{n-1}x^{n-1},$$

with $f_i \in \mathbb{Z}_p$. So there are p^n elements in $\mathbb{Z}_p[x]_m$. It does not matter which irreducible m we select, they will yield the same fields.

All the finite fields are formed this way, we denote them by $\mathbb{F}_q = GF(q)$ with $q = p^n$.

4.7 Shamair's secret sharing

Let us assume that $2 \leq k \leq n$. We want to share a secret between n member, which they can open only when at least k members are present.

Let us a fix a finite field $F = GF(q)$ with $q > n$ and $0 \neq \alpha \in F$ such that $\mathrm{ord}_q(\alpha) \geq n$. The secret they want to share and cannot open without k members of the party present is $s \in F$. An orchestrator must select $f_i \in_R F$ for $i = 1, \ldots, k-1$ and create the values $b_i = f(a_i)$ withe the polynomial

$$f = s + f_1x + \cdots f_{k-1}x^{k-1}$$

where $a_i = \alpha^i$ and $i = 1, \ldots, n$. The secret is $s = f(0)$. The orchestrator then distributes the the pairs (a_i, b_i) between the members $(i = 1, \ldots, n)$.

If the

$$\{i_1, i_2, \ldots, i_k\} \subset \{1, \ldots, n\}$$

members are present they can create the Lagrange interpolation polynomial g for which $g(a_{i_j}) = b_{i_j}$. Since $\deg(f), \deg(g) \leq k-1$ and they agree on k different points they must be the same polynomial, i.e. $f = g$. The members present can recover the secret $s = g(0)$.

4.8 Error correction with polynomials

4.8.1 Baby Reed-Solomon error correction

The symbols of the message are coming from the finite field $F = GF(q)$. The length of the message is k and we add m extra redundant symbols during encoding. The length of the encoded message is $n = k + m$. Select nonzero $\alpha \in F$ such that $\mathrm{ord}_q(\alpha) \geq n$.

Encoding The message $m = (m_0, \ldots, m_{k-1}) \in F^k$. Let us create the Lagrange interpolation polynomial g for which $g(\alpha^i) = m_i$ holds for $i = 0, \ldots, k-1$. The encoded message is

$$\begin{aligned} F \ni e &= (e_0, \ldots, e_{k-1}, e_k, \ldots, e_{n-1}) \\ &= (g(\alpha^0), \ldots, g(\alpha^{k-1}), g(\alpha^k), \ldots, g(\alpha^{n-1})) \\ &= (m_0, \ldots, m_{k-1}, g(\alpha^k), \ldots, g(\alpha^{n-1})). \end{aligned}$$

Decoding We receive $r = (r_0, \ldots, r_{n-1}) \in F^n$. For all k element index set

$$\{i_0, \ldots, i_{k-1}\} \subset \{0, \ldots, n-1\},$$

create the Lagrange interpolation polynomial g with $g(\alpha^{i_j}) = r_{i_j}$ for $j = 0, \ldots, k-1$ and add one vote for this polynomial.

If there was no error in transmission, only one polynomial is created and it gets all the votes. In this case, decoding is just trimming the redundancy symbols, i.e. $m_i = r_i$ for $i = 0, \ldots, k-1$.

If there was at least one error, different polynomials will be created. If there is a polynomial g^* which has the majority of votes, we can decode with it. Let $m_i = g^*(\alpha^i)$ for $i = 0, \ldots, k-1$. If we cannot select a winner, then we cannot correct the error and must report it.

4.9 Grownup Reed-Solomon error correction

Let $F = GF(q)$, $0 \neq \alpha \in F$ with $\mathrm{ord}_q(\alpha) \geq n$ and $0 < k < n$. We want to detect and correct errors during the transmission of $m = m_0 + m_1 x + \cdots m_{k-1} x^{k-1}$.

Encoding Let us define the generator polynomial of the encoding with $g = \prod_{i=1}^{t} \left(x - \alpha^i\right)$ where $t = n - k$. The encoded message is $c = mx^t - r$, where $r = mx^t \bmod g$.

Decoding Let us assume that we receive $c' = c + e$. Since $g \mid c$, every root of g must be a root of c as well. Let *syndrome polynomial* $s = s_0 + s_1 x + \cdots s_{t-1} x^{t-1}$ where $s_i = c'(\alpha^{i+1})$. If $e = 0$, then $s = 0$ and we can accept c' as a transmission without any error. We can extract the original message as the last k coefficients of c'.

If $s \neq 0$, then there was error during transmission, so $e \neq 0$. We can modify the extended Euclidean algorithm to produce the *error location polynomial* L and the *error value polynomial* E in Algorithm 9.

Algorithm 9 Calculating the error location and error value polynomials

procedure $\textsc{CalculateErrorPolynomails}(s,\ t)$
  $r^{(\text{old})},\ y^{(\text{old})} \leftarrow x^t,\ 0$
  $r^{(\text{new})},\ y^{(\text{new})} \leftarrow s,\ 1$
  while $\deg(r^{(\text{new})}) > \frac{t}{2}$ **do**
    $r^{(\text{old})},\ q,\ r^{(\text{new})} \leftarrow r^{(\text{new})},\ \textbf{divmod}\left(r^{(\text{old})}, r^{(\text{new})}\right)$
    $y^{(\text{old})},\ y^{(\text{new})} \leftarrow y^{(\text{new})},\ y^{(\text{old})} - q y^{(\text{new})}$
  end while
  $L \leftarrow y^{(\text{new})} / y^{(\text{new})}(0)$
  $E \leftarrow r^{(\text{new})} / y^{(\text{new})}(0)$
  return $L,\ E$
end procedure

It can be proven that $L = \prod_{e_j \neq 0} \left(1 - \alpha^j x\right)$. This means that if α^{-j} is a root of L, then $e_j \neq 0$, i.e. j is an error position of c'. For the other computed polynomial we have that $E = \sum_{e_j \neq 0} \alpha^j e_j L_j$ where

$$L_j = \prod_{\substack{e_i \neq 0 \\ j \neq i}} \left(1 - \alpha^i x\right).$$

Therefore

$$e_j = \frac{E(\alpha^{-j})}{\alpha^j L_j(\alpha^{-j})}$$

is the error at location j.

4.9.1 Example

Let $F = \mathbb{Z}_{11}$, $n = 10$, $k = 6$ so $t = n - k = 4$. In $\mathbb{Z}_1 1$ we have that $\mathrm{ord}_{11}(2) = 10$, so we can use $\prod_{i=1}^{4} \left(x - 2^i\right)$ for encoding. Note that every operator is done with respect to $\mod 11$. In particular, divisor is done by multiplication with the modular inverse.

Let us assume that we receive $c' = 4 + 4x + 8x^2 + 4x^3 + x^4 + 10x^5 + 10x^6 + 6x^7 + 2x^8$. To calculate the coefficients of the syndrome polynomial s we plug in the values 2, $2^2 = 4$, $2^3 = 8$ and $2^4 \mod 11 = 5$ into c'

$$\begin{aligned}
s_0 &= 4 + 4 \cdot 2 + 8 \cdot 2^2 + 4 \cdot 2^3 + 2^4 + 10 \cdot 2^5 + 10 \cdot 2^6 + 6 \cdot 2^7 + 2 \cdot 2^8 = 0, \\
s_1 &= 4 + 4 \cdot 4 + 8 \cdot 4^2 + 4 \cdot 4^3 + 4^4 + 10 \cdot 4^5 + 10 \cdot 4^6 + 6 \cdot 4^7 + 2 \cdot 4^8 = 10, \\
s_2 &= 4 + 4 \cdot 4 + 8 \cdot 8^2 + 4 \cdot 8^3 + 8^4 + 10 \cdot 8^5 + 10 \cdot 8^6 + 6 \cdot 8^7 + 2 \cdot 8^8 = 2, \\
s_3 &= 4 + 4 \cdot 5 + 8 \cdot 5^2 + 4 \cdot 5^3 + 5^4 + 10 \cdot 5^5 + 10 \cdot 5^6 + 6 \cdot 5^7 + 2 \cdot 5^8 = 5.
\end{aligned}$$

In summary $s = 10x + 2x^2 + 5x^3$.

During Algorithm 9 for the syndrome polynomial with $t = 2$, we stop the extended Euclidean algorithm steps at

$$r^{(2)} = 2x, \qquad y^{(2)} = 9 + 7x + 4x^2.$$

The inverse of $y^{(2)}(0) = 9$ is 5, so $L = 9x^2 + 2x + 1 = (1 - 4x)(1 - 5x)$. From this we can see that 4^{-1} and 5^{-1} are the roots of L. Since $4 = 2^2$ and $5 = 2^4$, we have that the locations of errors are $j = 2, 4$.

The error polynomial is $E = 5r^{(2)} = 10x$. Since $L_2 = 1-5x$ and $L_4 = 1-4x$, we can calculate the error values as

$$e_2 = \frac{E\left(4^{-1}\right)}{2^2 L_2\left(4^{-1}\right)} = \frac{8}{4 \cdot 8} = 3$$

and

$$e_4 = \frac{E\left(5^{-1}\right)}{2^4 L_4\left(5^{-1}\right)} = \frac{2}{5 \cdot 9} = 2,$$

so the error is $e = 3x^2 + 2x^4$.