

CS-E4500 Problem Set 4

Jaan Tollander de Balsch - 452056

February 9, 2019

(Gathen and Gerhard 2013, chaps. 5.1–5.4 and 10.1–10.3)

Problem 1

Let F be a finite field. In this case $F = \mathbb{Z}_p = \{0, 1, \dots, p\}$ for p prime. Let $\varphi_0 \in F$ be a **secret** that we'll split into s **shares** such that **knowledge** of any k shares enables recovery of the secret.

- 1) Let $\xi_1, \xi_2, \dots, \xi_s \in F$ be **distinct** and **nonzero**.
- 2) Select elements $\varphi_1, \varphi_2, \dots, \varphi_{k-1}$ independently and uniformly at random.
- 3) Let $f = \varphi_0 + \varphi_1 x + \varphi_2 x^2 + \dots + \varphi_{k-1} x^{k-1} \in F[x]$.
- 4) For $j = 1, 2, \dots, s$ share j is the pair $(\xi_j, f(\xi_j)) \in F^2$.

The secret can be **recovered** by interpolating the k shares back into polynomial f and evaluation the polynomial at $f(\xi_0) = f(0) = \varphi_0$. We'll use Lagrange interpolation as the interpolating algorithm.

```
from sympy import *
init_printing("mathjax")
import random

a = 0
b = 99
p = nextprime(b)
a, b, p

(0, 99, 101)

Z_p = list(range(p+1))
secret = random.randint(a, b)
shares = 10
knowledge = 5
secret, shares, knowledge
```

(42, 10, 5)

Step 1

```
xi = random.sample(Z_p[1:], shares)
xi
```

[3, 51, 85, 33, 7, 35, 81, 2, 1, 26]

Step 2

```
phi = random.sample(Z_p, knowledge-1)
phi
```

[17, 42, 50, 9]

Step 3

```
x = symbols('x', integer=True, positive=True)
f = Poly(reversed([secret]+phi), x)
f
```

$\text{Poly}(9x^4 + 50x^3 + 42x^2 + 17x + 42, x, \text{domain} = \mathbb{Z})$

Step 4

```
f_xi = [f(x)%p for x in xi]
f_xi
```

[25, 11, 30, 55, 73, 2, 42, 81, 59, 53]

```
pairs = list(zip(xi, f_xi))
pairs
```

[(3, 25), (51, 11), (85, 30), (33, 55), (7, 73), (35, 2), (81, 42), (2, 81), (1, 59)]

Recovering the secret

```
data = random.sample(pairs, knowledge)
data
```

[(81, 42), (3, 25), (1, 59), (7, 73), (85, 30)]

```
xi2, f_xi2 = list(zip(*data))
xi2, f_xi2
```

((81, 3, 1, 7, 85), (42, 25, 59, 73, 30))

Source: <https://stackoverflow.com/questions/4798654/modular-multiplicative-inverse-function-in-python/4798776>

```
def egcd(a, b):
    if a == 0:
        return (b, 0, 1)
    else:
        g, y, x = egcd(b % a, a)
        return (g, x - (b // a) * y, y)

def modinv(a, m):
    g, x, y = egcd(a, m)
    if g != 1:
        raise Exception('modular inverse does not exist')
    else:
        return x % m

def lagrange_interpolation(u, v):
    assert len(u) == len(v)
    n = len(u)
    P = 0
    for i in range(n):
        s = v[i]
        for j in range(n):
            if i == j:
                continue
            s *= (x - u[j]) * modinv((u[i] - u[j]) % p, p)
        P += s
    return poly(expand(P, modulus=p))

P = lagrange_interpolation(xi2, f_xi2)
P
```

$\text{Poly}(9x^4 + 50x^3 + 42x^2 + 17x + 42, x, \text{domain} = \mathbb{Z})$

Lets verify that our interpolation polynomial is correct.

f==P

True

Now we can recover the secret and verify that it is correct.

P(0)

P(0) == secret
True

Problem 2

Let R be a ring and let $q, r \in R[x]$ with $a = qb + r$ and $\deg r < \deg b$ where b monic.

(a)

Show that for all $\xi \in R$ and $f \in R[x]$ we have $f(\xi) = f \bmod (x - \xi)$.

Let $a = f$ and $b = (x - \xi)$ then

$$\begin{aligned} f(x) &= q(x) \cdot (x - \xi) + r(x) \\ f(\xi) &= q(\xi) \cdot (\xi - \xi) + r(\xi) \\ f(\xi) &= r(\xi) \\ f(\xi) &= f \bmod (x - \xi). \end{aligned}$$

(b)

Let $a, b, c \in R[x]$, with b and c monic, and suppose that c divides b . Show that $a \bmod c = (a \bmod b) \bmod c$.

We have equalities $a = q_1b + r_1$ and $b = q_2c + r_2$ where the remainders are

$$r_1 = a \bmod b$$

and $r_2 = 0$ since c divides b .

The remainder r_1 can also be written in this form $r_1 = q'c + r'$ where $\deg r' < \deg c$. Then we have a remainder

$$r' = r_1 \bmod c = (a \bmod b) \bmod c.$$

We also have

$$\begin{aligned} a &= q_1b + r_1 \\ a &= q_1(q_2c) + (q'c + r') \\ a &= (q_1q_2 + q')c + r'. \end{aligned}$$

Then the remainder r' has equivalency

$$r' = a \text{ rem } c.$$

By combining the equivalencies for the remainder r' we have

$$a \text{ rem } c = (a \text{ rem } b) \text{ rem } c.$$

Problem 3

Let R be a ring, let $\xi_0, \xi_1, \dots, \xi_{e-1} \in R$, and $\lambda_0, \lambda_1, \dots, \lambda_{e-1} \in R$ be given as input. The form of the Lagrange interpolation polynomial suggests that one should first seek to construct the coefficients of the polynomial

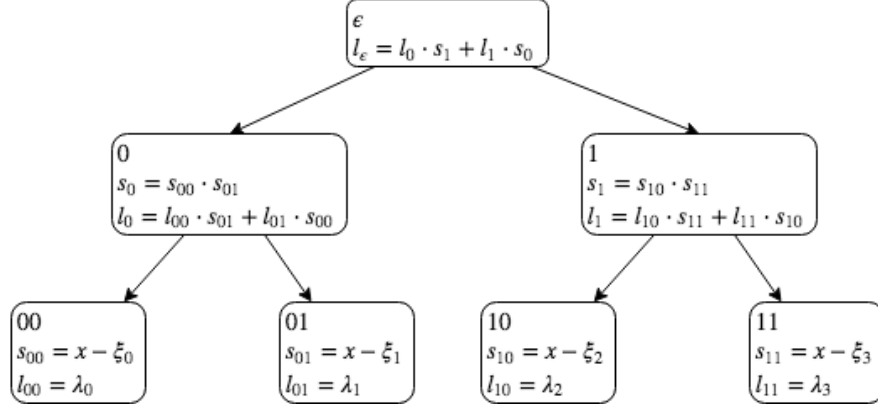
$$L_e(x) = \sum_{i=0}^{e-1} \lambda_i \prod_{j=0, j \neq i}^{e-1} (x - \xi_j) \in R.$$

Show that we can compute the coefficients of L in $O(M(e) \log e)$ operations in R . You may assume that $e = 2^k$ for a nonnegative integer k . Here $M(e) = e \log e \log \log e$.

The common substructure within the polynomials is evident if we expand the Lagrange polynomials with $e \in \{1, 2, 4\}$

$$\begin{aligned} L_1(x) &= \lambda_0 \\ L_2(x) &= \lambda_0(x - \xi_1) + \lambda_1(x - \xi_0) \\ L_4(x) &= \lambda_0(x - \xi_1)(x - \xi_2)(x - \xi_3) + \\ &\quad \lambda_1(x - \xi_0)(x - \xi_2)(x - \xi_3) + \\ &\quad \lambda_2(x - \xi_0)(x - \xi_1)(x - \xi_3) + \\ &\quad \lambda_3(x - \xi_0)(x - \xi_1)(x - \xi_2) \\ &= (\lambda_0(x - \xi_1) + \lambda_1(x - \xi_0)) \cdot (x - \xi_2)(x - \xi_3) + \\ &\quad (\lambda_2(x - \xi_3) + \lambda_3(x - \xi_2)) \cdot (x - \xi_0)(x - \xi_1). \end{aligned}$$

As can be seen, L_1 is the base case (leaf nodes) and L_2 (non-leaf nodes) form the rule for the recursive case. Then L_4 can be computed using these rules as seen on the binary tree representation.



The **generalized form**: Let the depths of the nodes in the binary tree starting from root be $i = 0, 1, \dots, k$ where $k = \log e$. We'll denote the nodes with binary string $\{0, 1\}^i = \{\{\varepsilon\}, \{0, 1\}, \{00, 01, 10, 11\}, \dots\}$ for depths $i = 0, 1, 2, \dots$. There are binary 2^i strings per at depth i , i.e. number of nodes at depth i .

TODO: the maximum size of polynomial at depth i

Each **leaf node** $v \in \{0, 1\}^k$

$$\begin{aligned} s_v &= x - \xi_v \\ l_v &= \lambda_v \end{aligned}$$

In each **non-leaf** node $u = \{0, 1\}^{k'}$ where $0 \leq k' < k$ the algorithm does the following computations

$$\begin{aligned} s_u &= s_{u0} \cdot s_{u1} \\ l_u &= l_{u0} \cdot s_{u1} + l_{u1} \cdot s_{u0}. \end{aligned}$$

The algorithm will terminate once the **root** node ε is reached. Then $l_\varepsilon = L(x)$. There is no need to calculate s_ε . Since $n = \deg s_u > \deg l_u$ for all $u \in \{0, 1\}^k$ the total computational complexity in each node is $O(n \log n)$ using **fast polynomial multiplication**.

Total computational complexity from the multiplication operations can be calculated multiplying the number of nodes m_i with the complexity of multiplying polynomials with maximum degree of n_i at depth i and summing over the total depth of the binary tree $i = 0, 1, \dots, k$. At depth i the binary tree has $m_i = 2^i$

nodes and polynomials have maximum degree of $n_i = 2^{\log e - i}$.

$$\begin{aligned}
& \sum_{i=0}^{\log e} m_i \cdot O(n_i \log n_i) \\
&= \sum_{i=0}^{\log e} 2^i \cdot O(2^{\log e - i} \log 2^{\log e - i}) \\
&= O\left(\sum_{i=0}^{\log e} 2^i \cdot 2^{\log e - i} \log 2^{\log e - i}\right) \\
&= O\left(\sum_{i=0}^{\log e} e(\log e - i)\right) \\
&= O\left(e \sum_{i=0}^{\log e} i\right) \\
&= O\left(e \cdot \frac{\log e(\log e + 1)}{2}\right) \\
&= O(e(\log e)^2).
\end{aligned}$$

Problem 4

Let R be a ring and let $\xi_0, \xi_1, \dots, \xi_{e-1} \in R$ and $\eta_0, \eta_1, \dots, \eta_{e-1} \in R$ such that $\xi_i - \xi_j$ is a unit in R for all $0 \leq i < j \leq e-1$. Show that we can compute the coefficients of the Lagrange interpolation polynomial

$$L(x) = \sum_{i=0}^{e-1} \left(\eta_i \prod_{j=0, j \neq i}^{e-1} (\xi_i - \xi_j)^{-1} \right) \prod_{j=0, j \neq i}^{e-1} (x - \xi_j) \in R[x]$$

that satisfies $L(\xi_i) = \eta_i$ for all $i = 0, 1, \dots, e-1$ in $O(M(e) \log e)$ operations in R . You may assume that $e = 2^k$ for a nonnegative integer k .

Let $f(x)$ be a polynomial

$$f(x) = \sum_{i=0}^{e-1} \lambda_i \prod_{j=0, j \neq i}^{e-1} (x - \xi_j),$$

where $\lambda_i = 1$ for all $i = 0, 1, \dots, e-1$. Then its coefficients can be calculated using the algorithm from problem 3 in $O(M(e) \log e)$ operations.

By batch evaluation the polynomial f in points ξ_k where $k = 0, 1, \dots, e-1$ we have

$$\begin{aligned} f(\xi_k) &= \sum_{i=0}^{e-1} \prod_{j=0, j \neq i}^{e-1} (\xi_k - \xi_j) \\ &= \prod_{j=0, j \neq k}^{e-1} (\xi_k - \xi_j). \end{aligned}$$

Batch evaluation can be done in $O(M(e) \log e)$ operations. The inverses of these values are the terms inside the Lagranges interpolation polynomial

$$\begin{aligned} f(\xi_k)^{-1} &= \left(\prod_{j=0, j \neq k}^{e-1} (\xi_k - \xi_j) \right)^{-1} \\ &= \prod_{j=0, j \neq k}^{e-1} (\xi_k - \xi_j)^{-1}. \end{aligned}$$

Using the results above, the Lagranges interpolation polynomial takes form

$$L(x) = \sum_{i=0}^{e-1} \lambda_i \prod_{j=0, j \neq i}^{e-1} (x - \xi_j)$$

where $\lambda_i = \eta_i f(\xi_i)^{-1}$ for all $i = 0, 1, \dots, e-1$. Then it's coefficients can be calculated using the algorithm from problem 3 in $O(M(e) \log e)$ operations.

Therefore the total amount of operations is $O(M(e) \log e)$.

References

Gathen, Joachim von zur, and Jurgen Gerhard. 2013. *Modern Computer Algebra*. 3rd ed. New York, NY, USA: Cambridge University Press.