Assignment 2

Bas van den Boom (1414879) Niels Gorter (1332678)

Marlene van Laar (1327518)

October 22, 2020

Contents

1 Basic polynomial arithmetic

We created a base class called Poly, which represents a polynomial. This poly class can be constructed by feeding it either the input from the exercise files, passing a string and a modulus, or a coefficient, a modulus and a degree from which it will construct a polynomial of a certain degree. The last two options were used for development and debugging purposes as they were very convenient. The class then stores an internal array containing all the coefficients and the modulus which is used to reduce all coefficients after operations.

1.1 Displaying a polynomial

In order to conveniently show a result, we need a way to display a polynomial. We implemented this by walking over the coefficients from high to low degree and by then appending the coefficient together with an X^n, where n is the degree. When a coefficient was zero, we did not append anything and when a coefficient is negative, we put a '-' sign in front. Finally we checked whether the string was empty (which happens with a zero polynomial), and if so returned "0".

1.2 Operating with a modulus

Since we are operating in $\mathbb{Z}/p\mathbb{Z}[X]$, where p is the modulus, we need to make sure that we stay in $\mathbb{Z}/p\mathbb{Z}[X]$. This can easily be done by applying the modulus to every coefficient of a polynomial. We implemented this by looping over the coefficients and using the python % operator on each coefficient. This operation was then called during or at the end of other operations, so the result would be in $\mathbb{Z}/p\mathbb{Z}[X]$. To prevent repetitiveness, we will not mention the use of this function in other parts, unless it is a key operation within an algorithm.

1.3 Addition and subtraction

Addition and subtraction were fairly easy to implement. By walking over the array of coefficients from right to left (so we start with degree 0 coefficients, then degree 1, etc.) and adding the coefficients together we got the sum of the two polynomials. All that remained was then applying the modulus (see the section Modulus above). For subtraction, we used the exact same method, but instead we used subtraction instead of addition.

1.4 Multiplication

When multiplying two polynomials f and g, things already get a bit more complicated. First, we create a new list of zero coefficients, with as length the sum of the orders of the polynomials + 1 (the +1 is there for the zero degree coefficient). We do this because we know that $deg(f * g) = deg(f) + deg(g)$. Then we use a variation of the primary school method of multiplication on the coefficients using a nested for loop looping over all coefficient pairs. All pairs get multiplied and then added to the new array of coefficients with as degree the sum of degrees of the individual coefficients.

1.5 Division and GCD

1.5.1 Long division (based on algorithm 2.2.6 from script)

First it is checked whether the divisor is the zero-polynomial or if the modulus is zero, which in this case an error is thrown.
For the quotient, an array of length equal to the dividend is initialised to zeroes and for the remainder, the dividend is copied onto a new list.

While the degree of the remainder is higher or equal than the degree of the divisor, the algorithm loops to find the coefficient that multiplied by the divisor eliminates the leading term of the dividend. It then calculates the quotient (by finding the correct position within the array and adding the leading coefficient) and the remainder (by subtracting the divisor multiplied by the quotient from the dividend). It then proceeds to divide the remainder by the divisor. In the end, it flips the array of the quotient and returns it alongside the remainder with leading zeroes removed.

1.5.2 Extended Euclidian algorithm (GCD) (based on algorithm 2.2.11 from script)

The algorithm closely follows the pesudocode provided in 2.2.11 from the script, with the exception that in the end it returns the GCD.

1.6 Modular equality

1.6.1 Modulo a polynomial

In case the modulo provided is an integer, the algorithm loops through the coefficients of the polynomial and reduces them (mod m).
If the modulo is the zero-polynomial, the algorithm simply returns the given polynomial.
If the degree of the polynomial is smaller than the degree of the modulus, the algorithm returns the polynomial with coefficients reduced modulo m.

1.6.2 Checking modular equality

Checking whether two polynomials f and g are congruent modulo polynomial h is simple. Just check whether $f \mod h$ is equal to $g \mod h$.

1.7 Inverse

Finding the inverse of a polynomial a in $Z/pZ/f(X)$ is quite simple (section 3.3 in the algebra script). We use the euclidean algorithm on (a, f). If the gcd turns out to be 1, the inverse can be retrieved from the first coefficient of the result. If the gcd is not equal to 1, a has no inverse.

1.8 Irreducibility

1.8.1 Checking irreducibility (based on algorithm 5.1.4 from script)

To check whether a polynomial f is irreducible, the algorithm loops through values $t = 1$ to the degree of f. If for any value of t in this range $X^{mt} - X^m$ ($q = m$) is equal to 1 (mod m), it is determined reducible $\rightarrow$ False is returned. Otherwise, it returns True $\rightarrow$ f is irreducible.

1.8.2 Finding irreducible polynomials (loosely based on algorithm 5.1.6 from the script)

```
def find_irred_step(poly: Poly, d) -> (Poly, bool):
    found_polys = []
    if poly.irreducible():
        found_polys.append(poly)
    if d == 0:
        return found_polys
    for n in range(0, poly.m):
        newPoly = poly.copy()
        newPoly.data[d] = n
        found = find_irred_step(newPoly, d - 1)
        if len(found) > 0:
            found_polys += found
    return found_polys
```

Figure 1: Recursive step to produce possible combinations of polynomials and test for irreducibility

To find an irreducible polynomial, the algorithm initializes an array of zeroes of length equal to the desired degree+1. It then recursively finds all irreducible polynomials of the desired degree, starting with all terms up to term degree - d already filled in, by exhausting all possible combinations of elements $0, 1, ..., m-1$ of length $d-1$ (in conjunction with all possible choices from elements $1, ..., m-1$ for the leading term) and appends them to a list. From this list, it checks whether the test polynomial is included for demonstrative purposes, and as finding a specific polynomial only by taking random polynomials could take much longer than simply finding all of them. If not needed, a random element can be picked and printed.

2 Basic finite field arithmetic

We created a class Field which holds a modulus m (integer) and a polynomial $f(X)$ (Poly), representing the field $\mathbb{Z}/m\mathbb{Z}[X]/(f(X))$.

2.1 Addition and subtraction

For addition and subtraction, the polynomials are added/subtracted according to the algorithms in Poly and are then reduced (mod *the polynomial of the field*).

2.2 Multiplication

The polynomials are multiplied according to the algorithm in Poly and are then reduced (mod *the polynomial of the field*).

2.3 Tables

Before we can generate the tables, we need to know which elements are in the field. To find these elements we treat a polynomial as a $mod * X$-radix representation of a number. If we then loop through all numbers from 0 to the order, we generate all polynomials in a field from small to big.

2.3.1 Addition table

Once we have the elements within the field, generating the addition field is easy. We can just loop through all the cells in the table and compute the unique representative of the sum of the polynomials belonging to this cell.

2.3.2 Multiplication table

The multiplication table is generated in almost the same way as the addition table, except that we use the product instead of the sum.

2.4 Inverse (based on algorithm 3.3.3 from the script)

The algorithm computes the GCD of the polynomial and the field polynomial. If the GCD equals 1, the algorithm returns the field polynomial (mod ax). Otherwise, it raises an assertion error.

2.5 Division

The algorithm first checks whether the divisor is 0, and if so, throws an error.
Both the divisor and the dividend are reduced (mod *polynomial of the field*). If the remainder of their division is $\neq$ zero, the field polynomial is added to the dividend. Otherwise, the quotient is returned.
If the remainder of the division between the new dividend and the divisor is $\neq 0$, the dividend becomes the initial dividend subtracted by the field polynomial. Otherwise, the quotient is returned.
If the remainder of the division between the new dividend and the divisor is $\neq 0$, the division is aborted. Otherwise, the quotient is returned.

2.6 Modular equality

An equality check is performed on the polynomials reduced (mod *polynomial of the field*).

2.7 Primitive elements

2.7.1 Checking if a element is primitive (based on algorithm 4.4.3 from script)

The distinct prime factors of the *order* $-1 = q - 1$ are obtained by looping through $2, .., i \leq \sqrt{n}$, checking if it divides $q - 1$ and if $i + 1$ doesn't, which in this case, it divides $q - 1$ by i and adds i as a prime factor. To remove duplicates, we made the elements of the prime factors list keys of a dictionary and added them to a new list. i is initialized to 0 because lists begin at index 0. From there, the algorithm closely follows the pseudocode provided in 4.4.3 from the script.

2.7.2 Finding primitive elements (based on algorithm 4.4.4 from the script)

Analogously to the irreducible finder from Poly, the algorithm produces random polynomials, reduces them (mod *polynomial of the field*) and checks whether it's a primitive element. It stops when it's exhausted all possibilities $\rightarrow$ when a generated polynomial equals the polynomial of the field.

3 User guide

In order to use our software, you need to have Python 3.6+ installed with the `asn1tools` library. One can then prepare an `input.ops` file according to the specification given in `operations.asn`. When this input file is placed in the same directory as `main.py`, the user can simply run `main.py` and the results of the operations in `input.ops` will be placed in a file `output.ops` in the same directory as `main.py`, they will also be printed to the console for quick visual checking.

4 Illustrative examples

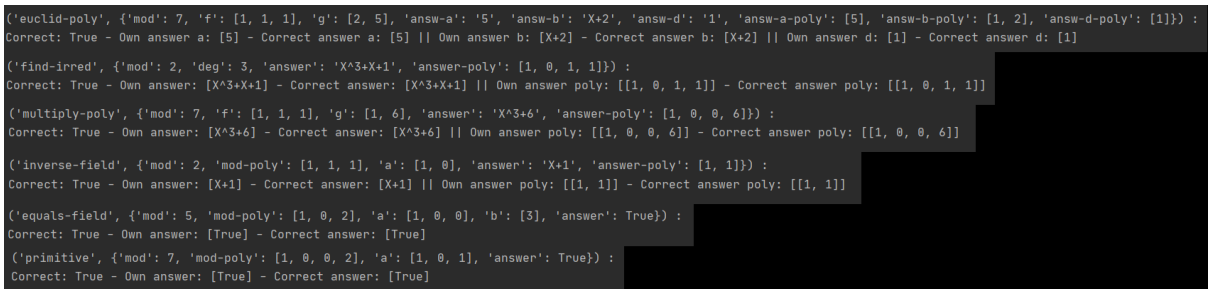

Figure 2: Examples

5 Limitations

All coefficients are expressed in 32-bit integers. This means that a coefficient must be at least -2147483648 and at most 2147483647. Furthermore we used the fact that the modulus p is at most 100 in a couple of places to justify the use of brute-force algorithms. Although there is no hard limit for the modulus in terms of correct functioning of the software, using a big modulus may lead to poor performance or even incorrect calculations if they use the list of known primes.

6 Contributions

Bas: Added answer checks, polynomial division, extended euclidean algorithm, irreducibility check, field base class, field addition, field subtraction, field equality, field division, field inverse, field primitivity check, display-field, addition and multiplication tables, comments, efficiency. Report sections 1, 1.1, 1.2, 1.3, 1.4, 2.3, 3, 5.

Niels: Answer checks, polynomial string representation, addition, subtraction, multiplication, long division, extended euclidian algorithm, modulo, modular equality, checking irreducibility, finding irreducible polynomials, field: checking primitivity, finding primitives, Poly class helper functions: power, modulo, power, gcd, $<$, $>$, $\leq$, $\geq$, $\neq$, $=$, list trimming, copying polynomial, leading coefficient, degree, length, function types, string interpretation, comments, efficiency. Report sections 1, 1.8.2, 3, 5.

Marlene: Added answer checks, irreducibility check, finding irreducible polynomials, field addition/subtraction/multiplication, primitivity check, finding primitive. Report sections 1.5, 1.6, 1.7, 1.8, 2.1, 2.2, 2.4, 2.5, 2.6, 2.7, 3.

Mick: No longer in our group.

7 List of References

- Lecture notes algebra
- Script algebra v08
- AANT-AFS v0.76