BinaryPolynomials

May 10, 2021

1 Extension Fields with Polynomials

1.1 Overview

The goal of this project was to create a relatively efficient, and easy to use, implementation of binary polynomials over finite fields.

1.2 What is a Finite Field?

a Finite Field is a set of elements with finite cardinality(a finite number of elements) and the operations of addition, subtraction, division, and multiplication are defined.

For Example, The Integers, Z, over a prime number, p, give the finite field Z_p which contains the elements $\{0, ..., p-1\}$

The general arithmetic in Z_p is generally pretty straight-forward:

Addition $1 + 12 \equiv 6 \, mod \, 7$

Multiplication and Division $4 * 6 = 24 \equiv 3 \, mod \, 7$

$4/3 = 4 * inverse(3) = 4 * 5 \equiv 6 \, mod \, 7$

Note about Inverses An inverse in Z_p for an element, x, is defined as the element,x^{-1}, where $x * x^{-1} \equiv 1 \, mod \, p$

1.3 Extension Fields

From the ideas we see above in Z_p we get extension fields. One important one is extension fields of Z_{2^q} especially over the polynomials.

1.3.1 From Z_p to F_{2^q}

In Z_p we used a prime modulus to create a finite field, but in the polynomials we need an analogue: an irreducible polynomial or one which can't be factored and is of degree q

Finding these irreducible polynomials is a relatively hard, but mostly solved problem. So, I simply created a table of some common ones to be used here. A good table of these can be found here: https://www.hpl.hp.com/techreports/98/HPL-98-135.pdf

1.4 Binary Polynomials

Binary Polynomials are all in extension fields of Z_2

Let's take the example of $x+1$ in F_4 using the irreducible polynomial x^2+x+1

1.4.1 Encoding

For the sake of efficieny we are going to encode both polynomials as a sequence of bits with the a 1 at position i representing a coefficient next to the ith term, so we think of a polynomial as $c_n * x^n + ... + c_2 * x^2 + c_1 * x + c_0 * 1$ and the corresponding bit string is $c_n, ..., c_2, c_1, c_0$

so $x+1$ will become $0b011$

1.4.2 Arithmetic

Once these polynomials are encoded as bit strings it allows us to do arithmetic with them in clever ways use a lot of bit manipulation.

Addition To add two polynomials, p_1 and p_2, we add their coefficients modulo 2 so add($x+1$,x) would equate to:

$(1+1\,mod\,2)*x+(1+0\,mod\,2)*1 \;=\; 0*x+1*1 \;=1$ then to make sure we remain in the field we have to take the result, x, modulo the irreducible polynomial of the field -> $1\,mod\,x^2+x+1 \;=1$

so if we converted $x+1$ and x to the bit strings $0b11$ and $0b010$ respectively. To add these we can use the bitwise xor operation. by xor-ing these we get $0b001$ which encodes the polynomial 1

Multiplication We can multiply any p_1 and p_2 using another binary algorithm. for each '1' in p_2 we shift p_1 by the index of the '1' and then sum all of those together to get the result.

to multiply $0b010$ and $0b101$ on paper we can do something like this:

$$010 * 101 ---\;\; 01000 + 00010 ---{-}01010$$

Then, we take the result $0b01010$ mod x^2+x+1 to stay in F_4 which brings us to the modulo algorithm for polynomials

Modulo the goal of the modulo operation for a polynomial, $p(x)\;mod\;q(x)$, where $q(x)$ is an irreducible polynomial for the field we are in, is to bring $p(x)$ into one of the residue classes for the Field so it's degree must be less than that of the irreducible polynomial.

so, if we have $0b01010$ mod $0b111$ in F_4 on paper we would do:

$$01010 + 1110 ----\;\; 0110 + 111 ---{-}001$$

so, $0b01010$ mod $0b111 = 0b1$ in F_4

for this algorithm we shift the irreducible polynomial over to be the same degree as the dividend and then add them until the dividend has a smaller degree than the divisor.

Exponentiation and PowMod Although we could just use the ideas we have just learned to do $p(x)^a \, mod \, q(x)$ by multiplying $p(x)$ a times and then taking the remainder, we will not because it is rather time and memory ineficient.

Instead, we will be taking advantage of the Power Mod Algorithm which is much more efficient and adapting it to polynomials(Wikepedia link here: https://en.wikipedia.org/wiki/Modular_exponentiation#:~:text=The%20operation%20of%20modular%20expone

The pseudo code is:

- base - polynomial
- exp - Power
- p - Irreducible Polynomial
- b - accumulator polynomial

```
pow_mod( base, exp, p, b){

    if(exp==1){

        return base*b % p;

    }else if(exp%2==0){

        return pow_mod(base*base % p, exp/2, p, b);

    }

    return pow_mod(base*base % p, (exp-1)/2, p, base*b % p);

}
```

Inverse Just like we had inverses in Z_n we have multiplicative inverse of polynomials which can be computed as $p(x)^{order-2} mod q(x)$ where order is the order of the Field or the number of elements it contains.

Square Roots A similar "trick" can be done to compute square roots for an element in F_{2^q} by doing $p(x)^{2^{q-1}} \, mod \, q(x)$