Finite field arithmetic

From Wikipedia, the free encyclopedia

Arithmetic in a finite field is different from standard integer arithmetic. There are a limited number of elements in the finite field; all operations performed in the finite field result in an element within that field.

While each finite field is itself not infinite, there are infinitely many different finite fields; their number of elements (which is also called cardinality) is necessarily of the form p^n where p is a prime number and n is a positive integer, and two finite fields of the same size are isomorphic. The prime p is called the characteristic of the field, and the positive integer n is called the dimension of the field over its prime field.

Finite fields are used in a variety of applications, including in classical coding theory in linear block codes such as BCH codes and Reed–Solomon error correction and in cryptography algorithms such as the Rijndael encryption algorithm.

Contents

Effective polynomial representation

The finite field with p^n elements is denoted GF(p^n) and is also called the **Galois Field**, in honor of the founder of finite field theory, Évariste Galois. GF(p), where p is a prime number, is simply the ring of integers modulo p. That is, one can perform operations (addition, subtraction, multiplication) using the usual operation on integers, followed by reduction modulo p. For instance, in GF(5), 4+3=7 is reduced to 2 modulo 5. Division is multiplication by the inverse modulo p, which may be computed using the extended Euclidean algorithm.

A particular case is GF(2), where addition is exclusive OR (XOR) and multiplication is AND. Since the only invertible element is 1, division is the identity function.

Elements of GF(p^n) may be represented as polynomials of degree strictly less than n over GF(p). Operations are then performed modulo R where R is an irreducible polynomial of degree n over GF(p), for instance using polynomial long division. The addition of two polynomials P and Q is done as usual; multiplication may be done as follows: compute $W = P.Q$ as usual, then compute the remainder modulo R (there exist better ways to do this).

When the prime is 2, it is conventional to express elements of GF(p^n) as binary numbers, with each term in a polynomial represented by one bit in the corresponding element's binary expression. Braces ("{" and "}") or similar delimiters are commonly added to binary numbers, or to their hexadecimal equivalents, to indicate that the value is an element of a field. For example, the following are equivalent representations of the same value in a characteristic 2 finite field:

Polynomial: $x^6 + x^4 + x + 1$
Binary: {01010011}
Hexadecimal: {53}

Addition and subtraction

Addition and subtraction are performed by adding or subtracting two of these polynomials together, and reducing the result modulo the characteristic.

In a finite field with characteristic 2, addition modulo 2, subtraction modulo 2, and XOR are identical. Thus,

Polynomial: $(x^6 + x^4 + x + 1) + (x^7 + x^6 + x^3 + x) = x^7 + x^4 + x^3 + 1$
Binary: {01010011} + {11001010} = {10011001}
Hexadecimal: {53} + {CA} = {99}

Notice that under regular addition of polynomials, the sum would contain a term $2x^6$, but that this term becomes $0x^6$ and is dropped when the answer is reduced modulo 2.

Here is a table with both the normal algebraic sum and the characteristic 2 finite field sum of a few polynomials:

p_1	p_2	$p_1 + p_2$ (normal algebra)	$p_1 + p_2$ in GF(2^n)
$x^3 + x + 1$	$x^3 + x^2$	$2x^3 + x^2 + x + 1$	$x^2 + x + 1$
$x^4 + x^2$	$x^6 + x^2$	$x^6 + x^4 + 2x^2$	$x^6 + x^4$
$x + 1$	$x^2 + 1$	$x^2 + x + 2$	$x^2 + x$
$x^3 + x$	$x^2 + 1$	$x^3 + x^2 + x + 1$	$x^3 + x^2 + x + 1$
$x^2 + x$	$x^2 + x$	$2x^2 + 2x$	0

Note: In computer science applications, the operations are simplified for finite fields of characteristic 2, also called GF(2^n) Galois fields, making these fields especially popular choices for applications.

Multiplication

Multiplication in a finite field is multiplication modulo an irreducible reducing polynomial used to define the finite field. (I.e., it is multiplication followed by division using the reducing polynomial as the divisor—the remainder is the product.) The symbol "•" may be used to denote multiplication in a finite field.

Rijndael's finite field

Rijndael uses a characteristic 2 finite field with 256 elements, which can also be called the Galois field **GF**(2^8). It employs the following reducing polynomial for multiplication:

$$x^8 + x^4 + x^3 + x + 1.$$

For example, {53} • {CA} = {01} in Rijndael's field because

$(x^6 + x^4 + x + 1)(x^7 + x^6 + x^3 + x) =$

$(x^{13} + x^{12} + x^9 + \mathbf{x^7}) + (x^{11} + x^{10} + \mathbf{x^7} + x^5) + (x^8 + \mathbf{x^7} + x^4 + x^2) + (\mathbf{x^7} + x^6 + x^3 + x) =$

$x^{13} + x^{12} + x^9 + x^{11} + x^{10} + x^5 + x^8 + x^4 + x^2 + x^6 + x^3 + x =$

$x^{13} + x^{12} + x^{11} + x^{10} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + x^2 + x$

and

$x^{13} + x^{12} + x^{11} + x^{10} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + x^2 + x$ modulo $x^8 + x^4 + x^3 + x^1 + 1 =$ (11111101111110 mod 100011011) = {3F7E mod 11B} = {01} = 1 (decimal), which can be demonstrated through long division (shown using binary notation, since it lends itself well to the task. Notice that exclusive OR is applied in the example and not arithmetic subtraction, as one might use in grade-school long division.):

```
                 _______________
 11111101111110 (mod) 100011011
^100011011
 _____________
  1110000011110
 ^100011011
 _____________
   1101101011110
  ^100011011
  _____________
    101011101110
   ^100011011
   _____________
     01000110010
    ^100011011
    _____________
      00000001
```

(The elements {53} and {CA} are multiplicative inverses of one another since their product is 1.)

Multiplication in this particular finite field can also be done using a modified version of the "peasant's algorithm". Each polynomial is represented using the same binary notation as above. Eight bits is sufficient because only degrees 0 to 7 are possible in the terms of each (reduced) polynomial.

This algorithm uses three variables (in the computer programming sense), each holding an eight-bit representation. **a** and **b** are initialized with the multiplicands; **p** accumulates the product and must be initialized to 0.

At the start and end of the algorithm, and the start and end of each iteration, this invariant is true: **a** **b** + **p** is the product. This is obviously true when the algorithm starts. When the algorithm terminates, **a** or **b** will be zero so **p** will contain the product.

- Run the following loop eight times (once per bit). It is OK to stop when **a** or **b** are zero before an iteration:
    1. If the rightmost bit of **b** is set, exclusive OR the product **p** by the value of **a**. This is polynomial addition.

2. Shift **b** one bit to the right, discarding the rightmost bit, and making the leftmost bit have a value of zero. This divides the polynomial by **x**, discarding the x^0 term.
3. Keep track of whether the leftmost bit of **a** is set to one and call this value **carry**.
4. Shift **a** one bit to the left, discarding the leftmost bit, and making the new rightmost bit zero. This multiplies the polynomial by **x**, but we still need to take account of **carry** which represented the coefficient of x^7.
5. If **carry** had a value of one, exclusive or **a** with the hexadecimal number `0x1b` (00011011 in binary). `0x1b` corresponds to the irreducible polynomial with the high term eliminated. Conceptually, the high term of the irreducible polynomial and **carry** add modulo 2 to 0.

- **p** now has the product

This algorithm generalizes easily to multiplication over other fields of characteristic 2, changing the lengths of **a**, **b**, and **p** and the value `0x1b` appropriately.

Multiplicative inverse

The multiplicative inverse for an element **a** of a finite field can be calculated a number of different ways:

- By multiplying **a** by every number in the field until the product is one. This is a Brute-force search.
- For $n = 1$ one can take advantage of the analog of Fermat's little theorem, $a^{p-1} \equiv 1$ (for $a \neq 0$), thus the inverse of a is a^{p-2}
- By using the Extended Euclidean algorithm
- By making a logarithm table of the finite field, and performing subtraction in the table. Subtraction of logarithms is the same as division.

Program examples

C programming example

Here is some C code which will add, subtract, and multiply numbers in Rijndael's finite field:

```
/* Add two numbers in a GF(2^8) finite field */
uint8_t gadd(uint8_t a, uint8_t b) {
        return a ^ b;
}

/* Subtract two numbers in a GF(2^8) finite field */
uint8_t gsub(uint8_t a, uint8_t b) {
        return a ^ b;
}

/* Multiply two numbers in the GF(2^8) finite field defined
 * by the polynomial x^8 + x^4 + x^3 + x + 1 */
uint8_t gmul(uint8_t a, uint8_t b) {
        uint8_t p = 0;
        uint8_t counter;
        uint8_t carry;
        for (counter = 0; counter < 8; counter++) {
                if (b & 1)
```

```
                p ^= a;
            carry = (a & 0x80);
            a <<= 1;
            if (carry)
                a ^= 0x011D; /* what x^8 is modulo x^8 + x^4 + x^3 + x^2 + 1 */
            b >>= 1;
        }
        return p;
}
```

D programming example

This D program will multiply numbers in Rijndael's finite field and generate a PGM image:

```
/**
Multiply two numbers in the GF(2^8) finite field defined
by the polynomial x^8 + x^4 + x^3 + x + 1.
*/
ubyte gMul(ubyte a, ubyte b) pure nothrow {
    ubyte p = 0;

    foreach (immutable ubyte counter; 0 .. 8) {
        if (b & 1)
            p ^= a;
        immutable ubyte carry = a & 0x80;
        a <<= 1;
        if (carry != 0)
            a ^= 0b1_0001_1101; // x^8 + x^4 + x^3 + x^2 + 1.
        b >>= 1;
    }

    return p;
}

void main() {
    import std.stdio, std.conv;
    enum width = ubyte.max + 1, height = width;

    auto f = File("rijndael_finite_field_multiplication.pgm", "wb");
    f.writefln("P5\n%d %d\n255", width, height);
    foreach (immutable y; 0 .. height)
        foreach (immutable x; 0 .. width) {
            immutable char c = gMul(x.to!ubyte, y.to!ubyte);
            f.write(c);
        }
}
```

Arithmetic example (where the characteristic is not 2)

Addition: (A + B) mod Characteristic
Subtraction: (A - B) mod Characteristic
Multiplication: (A * B) mod Characteristic

This code is vulnerable to timing attacks when used for cryptography.

External links

- A description of Rijndael's finite field (http://www.samiam.org/galois.html)
- Galois Field Arithmetic Library C++ (http://www.partow.net/projects/galois/index.html)