

密码学实验二实验报告

姓名：孟衍璋 学号：16337183

一、 实验目的

在 $F_{2^{131}}$ 有限域下，实现一些基本运算——加法、求模、乘法、平方和求逆。

二、 实验内容

利用自己的学号，进行求逆，然后输出计算值。

如：

学号为 2，即表示为： x （即：10 二进制），然后计算 x 的逆为： $x^{130} + x^{12} + x + 1$

（即：11000000 00001000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 001 二进制）

由左到右，依次为 $1, x, x^2 \dots x^{130}$

三、 实验及算法原理

我在做本实验的过程中，主要是通过阅读助教发给我们的那篇论文与给出的实验参考的 PPT 才理解了该如何做。所以代码中的算法是复现了论文中的对应算法。

在实现中我使用了 `bitset` 结构，它可以用来进行一些状态储存的操作，类似于一个标记数组。移位或者异或运算比较方便。

加法部分：

只需要将对应输入的串最低位对齐，依次做按位异或运算。函数输入是

string，转换为 bitset 之后做按位异或，再转换为 string 输出。

求模部分：

将 $f(x)$ 转为对应的二进制表示形式，然后把输入按 64 位分组（不足的在右边补零）。之后进行一系列的移位与异或操作，得到结果转换为 string 输出。

乘法部分：

假设相乘的多项式是 a 与 b ，得到的结果为 c 。如果 a 的最低项次数为 1，则将 b 的值赋给 c ；如果 a 的最低项次数为 0，则将 c 全部置 0。之后遍历 a 的每一位， b 的值更新为左移一位后模 $f(x)$ 的结果，如果 $a[i]$ 为 1，将 c 的值更新为 $c+b$ ，这里可以调用之前的加法操作。结束之后输出 c 即为结果。

平方部分：

直接调用乘法运算，其中两个系数为要平方的多项式。

求逆部分：

首先要会求两个多项式的最大公因式的操作，这里采用欧几里得算法。之后便利用 $\gcd(a, f) = 1$ ，求出输入 a 的逆。

四、 实验代码

[illegible]

```

46     bitset<64> c0 (cx[0]);
47     bitset<64> c1 (cx[1]);
48     bitset<64> c2 (cx[2]);
49     bitset<64> c3 (cx[3]);
50     bitset<64> c4 (cx[4]);
51     bitset<64> t;
52     //c4 mod f(x)
53     t = c4;
54     c1 = c1 ^ (t << 63) ^ (t << 62) ^ (t << 61);
55     c2 = c2 ^ (t << 10) ^ (t >> 1) ^ (t >> 2) ^ (t >> 3);
56     //c3 mod f(x)
57     t = c3;
58     c0 = c0 ^ (t << 63) ^ (t << 62) ^ (t << 61);
59     c1 = c1 ^ (t << 10) ^ (t >> 1) ^ (t >> 2) ^ (t >> 3);
60     c2 = c2 ^ (t >> 54);
61     //c2
62     bitset<64> t1 (0xffffffffffffffff8);
63     bitset<64> t2 (0x0000000000000007);
64     t = c2 & t1;
65     c0 = c0 ^ (t << 10) ^ (t >> 1) ^ (t >> 2) ^ (t >> 3);
66     c1 = c1 ^ (t >> 54);
67     c2 = c2 & t2;
68
69     string m = c0.to_string();
70     m += c1.to_string();
71     m += c2.to_string();
72     //cout << "modulo:" << endl << m << endl;
73     return m;
74 }

```

```

76 string polynomial_multiply(string str_a,string str_b)
77 {
78     bitset<260> a (str_a);
79     bitset<260> b (str_b);
80     bitset<260> c;
81     if(a[str_a.size() - 1] == 1)
82     {
83         bitset<260> cc (str_b);
84         c = cc;
85     }
86     for (int i = 1; i < 130; ++i)
87     {
88         string temp_b = polynomial_modulo((b << 1).to_string());
89         bitset<260> t (temp_b);
90         b = t;
91         if(a[i] == 1)
92         {
93             string temp_c = polynomial_addition(b.to_string(),c.to_string());
94             bitset<260> tt (temp_c);
95             c = tt;
96         }
97     }
98
99     string str_c = c.to_string();
100     //cout << "multiply:" << endl << str_c << endl;
101     return str_c;
102 }

```

[illegible]

```

150 int main(int argc, char const *argv[])
151 {
152     //多项式最高位在右边
153
154     //131位
155     string add_a = "000101010010101010010101110111010101010101010000001111110101
156 0101010101010001010100101010101000001110101011101010101010101010";
157     string add_b = "01010101001010111111010101010100101010000001011111101010100
158 1010101000000111111001001010101010101001010101001000000111111110";
159     string add_c = polynomial_addition(add_a,add_b);
160     cout << "addition:" << endl << cut(add_c,129) << endl;
161     cout << endl;
162
163     //260位
164     string modulo_a = "0110101010011010101001101010100110101010011010101001101010100
165 1101010100110101001101010100110101001101010100110101010011010100110101010011010100
166 11010101001101010100110101010011010101001101010100110101010011010101001101010100
167 110101010011010101001101010100110101010";
168     string modulo_b = polynomial_modulo(modulo_a);
169     cout << "modulo:" << endl << cut(modulo_b,modulo_b.find_first_of("1")) << endl;
170     cout << endl;
171
172     //131位
173     string multiply_a = "101001010101001000000001011111100101010010101010000101010100
174 101010010101010010010101110010101000101010100101010010101001010100101010010";
175     string multiply_b = "010001010100010101010100101010101111111010101001000101010
176 100101001010000111010101111010101011000000101010111101001010100000011";
177     string multiply_c = polynomial_multiply(multiply_a,multiply_b);
178     cout << "multiply:" << endl << cut(multiply_c,multiply_c.find_first_of("1")) << endl;
179     cout << endl;
180
181     //131位
182     string square_a = "101001010010101010111111100101010100100010010100101010010101
183 010101010101001010101010101010010101010010101010010101001010101010101";
184     string square_b = polynomial_square(square_a);
185     cout << "square:" << endl << cut(square_b,square_b.find_first_of("1")) << endl;
186     cout << endl;
187
188     //16337183 -> 111110010100100100011111
189     string inverse_a = "111110010100100100011111"; //低位在右边
190     string inverse_b = polynomial_inverse(inverse_a);
191     cout << "inverse:" << endl << cut(inverse_b,inverse_b.find_first_of("1")) << endl;
192     cout << endl;
193
194     system("pause");
195     return 0;
196 }

```

五、 运行截图

```
C:\WINDOWS\system32\cmd.exe

addition:
010000000110000001101000111011111110000000101011011110010100001
11111110101011010110110000000001010010011111011111000101011010
10100

modulo:
1100101111110110111111011011111101101111100001111111011010001
101010010010001001001000100100100010010010001001001000011001101
00000000000000000000000000000000000000000000000000000000000000
10

multiply:
10000010001000011100001110111001111011111110010100111001111000
1000001011111111110001011110000110000111001001110000001111001000
110101000011101010111101010101011000000101010111101001010100000
110

square:
110011001101010100111011111101010100111101001000110101101111100
011111010001110111110101010010101000101011110110001111001011100
1010101001010101001010101001010101001010101001010101010101010
101

inverse:
100110000101011010001111101111001000000101111001000100100001011
111010100011010111011101000101111001101011010000000111011100101
01101

请按任意键继续. . .
```

六、 结果分析

输出依次是加法、求模、乘法、平方、求逆运算的结果。其中求逆运算的输入是将自己的学号转为二进制之后输入进去得到的结果。

七、 总结

这次的算法部分都是根据论文里给出的算法来完成的。主要的挑战就是看懂论文，其实看了之后实现起来也相对不是那么复杂。我用了 `bitset` 的结构，在实验过程中因为申请的空间不够大，导致赋值不成功，所以求逆运算一直有很大的问题，后来发现这个问题之后改大空间之后就解决了。

这次的实验让我对有限域的概念有了进一步了解，实际操作之后理解更

加深刻了。