实验二 有限域运算

> **17341125 蒙亚愿 信息安全**

一、实验目的

> 在深入了解了有限域的基础上，构造一个有限域，并在有限域之内进行运算。

二、实验要求

> 1、构造有限域GF(2^{127})：不可约多项式为p(x)=x^{127}+x+1；
>
> 2、给出GF(2^{127})上的加法、乘法、平方以及求逆的子程序；
>
> 3，将自己的学号转为二进制字符串，看作有限域中的元素a，计算元素的逆元素以及$a^{20190911}$
>
> 注：二进制的学号每一位数字用四位二进制比特串表示，8位学号表示为32位二进制串。如12345678-> 00010010000110100010101100111101000

三、实验介绍

3.1 实验综述

已知在$GF(2^{127})$中的不可约多项式$p(x)=x^{127}+x+1$，就可以通过不可约多项式构造一个次数小于127，系数$a_i \in \{0,1\}$的多项式集，其中所有的元素运算都是在$mod\ p(x)$下进行的。

加法：对于二进制数a, b每个元素都进行异或，或者说是$mod\ 2$加法，所以说没有进位，逐位运算。

乘法：对于二进制数a, b相乘，对于b中的每一位，若$b[i]=1$，a就左移i位，并与原来的a相加，结果$mod\ p(x)$。

平方：对于二进制数a，若$a[i]=1$，$answer[2i]=1$，结果$mod\ p(x)$

求逆：利用贝组定理、拓展欧几里得的算法计算，当a等于0时，即得逆。

计算$a^{20190911}$可用平方乘算法，最后得出的结果需要模$p(x)$。

3.2 开发环境

运行环境：Windows

使用语言：C++

四、算法原理

定义$p(x)=x^{127}+x+1$，$r(x)=x+1$，$M=128$。

4.1 加法运算

设二进制数a, b:

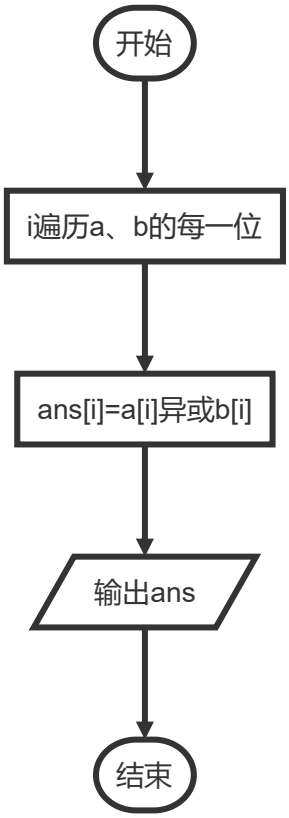
$$a = a_{M-1}x^{M-1} + a_{M-2}x^{M-2} + \ldots\ldots + a_1 x + a_0$$

$$b = b_{M-1}x^{M-1} + b_{M-2}x^{M-2} + \ldots\ldots + b_1 x + b_0$$

则结果$ans = (a_{M-1} \bigoplus b_{M-1})x^{M-1} + \ldots\ldots + (a_1 \bigoplus b_1)x + (a_0 \bigoplus b_0) = a\text{^}b$

即逐位进行异或运算得出结果。在a和b的次数都没有超过M时，ans的次数也不会超过M，不用模$p(x)$。

流程图:



4.2 乘法运算

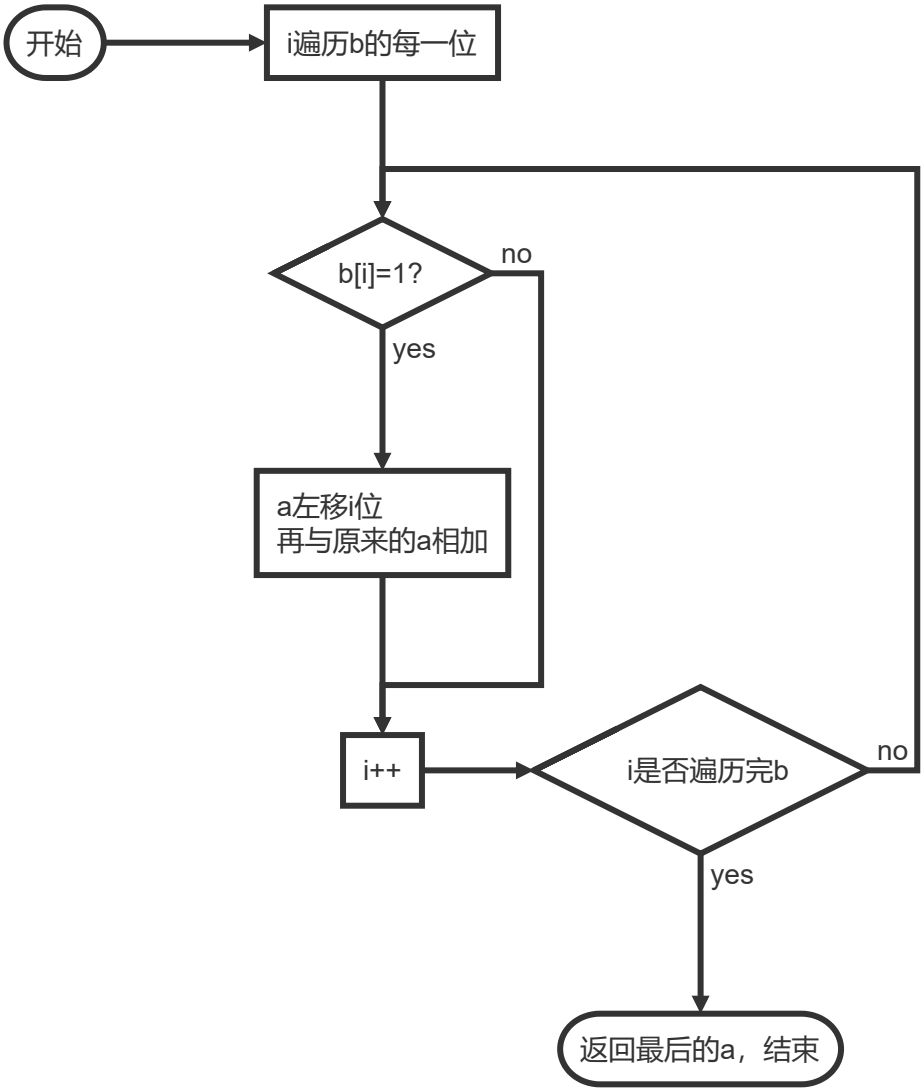
$$a = \sum_{i=0}^{M-1} a_i x^i$$

$$b = \sum_{i=0}^{M-1} b_i x^i$$

$$a * b = \sum_{i=0}^{M-1} a * b_i x^i$$

每乘一个b_i，就相当于左移i位，然后再与原来的a相加。

流程图:



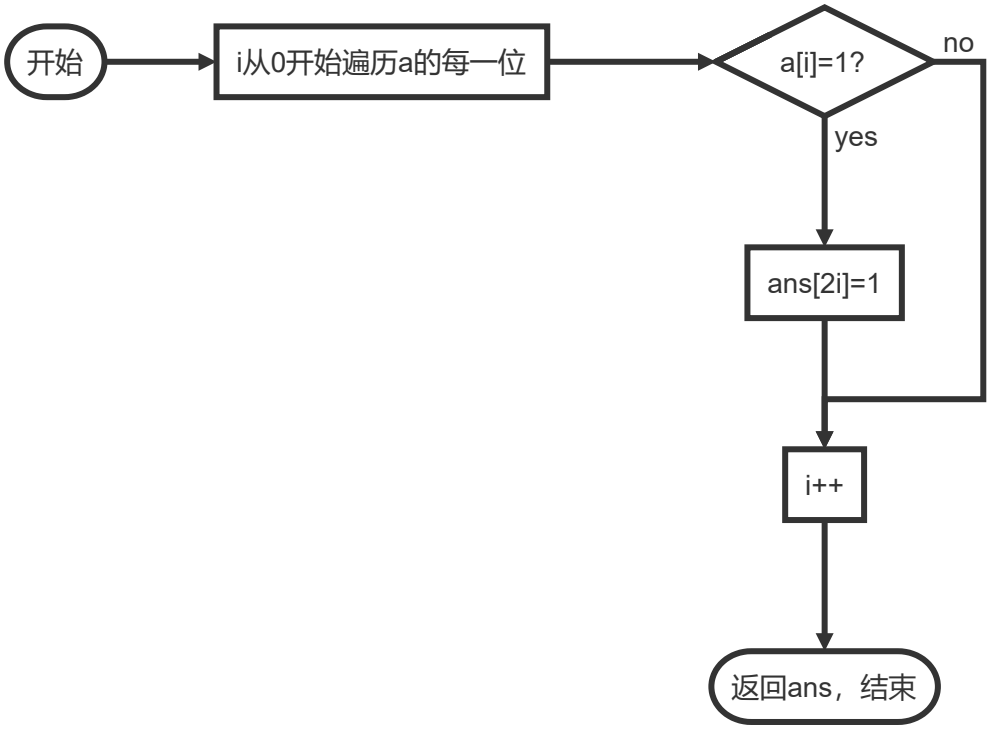
4.3 平方运算

经过手动验证，在进行平方运算时，若第i位的系数是1，则答案的第$2i$位是1。

$$a = \sum_{i=0}^{M-1} a_i x^i$$

$$a^2 = \sum_{i=0}^{M-1} a * a_i * x^i = \sum_{i=0}^{M-1} a_i x^{2i}$$

流程图：



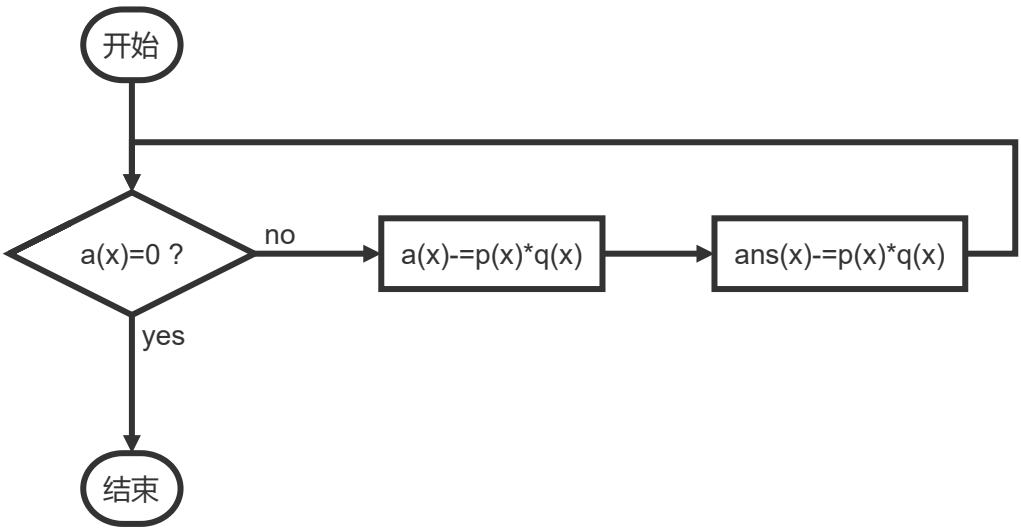
4.4 求逆运算

利用拓展欧几里得原理存在$q(x)=x^{|a(x)|-|p(x)|}$和$0<r(x)<p(x)$，$a(x)=p(x)*q(x)+r(x)$计算：

$$ans(x)=1$$
$$a(x)=a(x)-p(x)*q(x)$$
$$ans(x)=ans(x)-q(x)*p(x)$$

辗转相除法，计算直到$a(x)=0$,就得到ans。

流程图：



4.5 幂运算

利用平方乘算法，先把大指数20190911转成二进制数$e(x)$，长度为n，$ans = 1_{(2)}$，从高位起的每一位，ans都自己平方运算，若$e(i)$等于1，ans再乘底数a；其中，每一次运算，都是在$mod\ p(x)$下进行的。

```
// 平方乘算法
ans <- 1
e(x)= e.transformToBinary()
for i n to 0 do
    ans = ans*ans (mod p(x))
    if e(i)==1
        then ans = ans*a (mod p(x))
end
```

4.6 学号运算

把学号id看作有限域中的元素，就是先把学号转成二进制a，$a \mod p(x)$就可以化成有限域中的元素，稍后会解说其原理。而计算其指数，也就用到了以上所题的幂运算。

4.7 其他运算

4.7.1 求最高次数

计算多项式（二进制）的最高次数，即从最高位开始，返回去寻找第一个1的位数，即为最高次数。

```
// 求a(x)最高次数
for i M-1 to 0 do
    if a[i]==1
        then return i
end
```

4.7.2 模$p(x)$

要构造$GF(2^{127})$的有限域，通过模不可约多项式$p(x)$，使得所有的多项式的最高次数都不超过127。

对于任何多项式$a(x)$，都有以下两种情况：

1. 最高次数$|a(x)| < |p(x)| = 127$

   直接返回$a(x)$

2. 最高次数$|a(x)| \leq |p(x)|$

   $ans(x)$为$a(x)$中次数小于127的所有项的和。

   对于次数$\geq$127的每一项$a_i x^i$（且$a_i = 1$），都可以等于$a_i x^{i-127} * r(x)$，这样一来，其次数就会小于127。所以，遍历$a(x)$中次数大于等于127的每一项，都用以上方法替换$a_i x^i$，得到的结果在与原来的$ans(x)$模2相加，每次遍历完都检查是否$ans(x)$的次数小于127，是的话返回$ans(x)$，否则像刚才一样继续遍历替换。

```
// mod p(x)运算
if degree(a)<127
    then return a
    else
        while degree(a)>=127  do
                ans = a.str(M)
                for i 2*M to 0 do
                    if a[i]==1
                        then ans += (r<<(i-127))
                end
         end
```

五、代码设计及实现

5.1 main模块

主要用于初始化全局变量$p(x)=x^127+x+1$，$r(x)=x+1$，以及作为各个程序的入口。

```
#define MAX 127
bitset<MAX+1> px; //全局变量
bitset<MAX> rx; //长度为128
```

```
int main(){
    px[127] =px[0]=px[1]= 1;
    rx[0] = rx[1] = 1;
    cout<<"px="<<px<<"\nrx="<<rx<<endl<<endl;
//  for(int i=MAX-1;i>=0;i--)  if(px.test(i))  cout<<"x^"<<i<<"+";
     while(1){
        system("cls");
        cout<<"\n\n 请选择你要进行的计算："<<endl;
        cout<<"\n 1-加法     2-乘法     3-平方     4-求逆     5-学号计算     6-退出"
<<endl;
        cout<<"\n choice: ";
        int choice;
        cin>>choice;
        switch(choice){
            case 1:{
                system("cls");
                cout<<"\n\n 加法"<<endl;
                add();
                system("pause") ;
                break;
            }
            case 2:{
                system("cls");
                cout<<"\n\n 乘法"<<endl;
                bitset<MAX> a,b;
                cout<<"\n 请输入两个乘数："<<endl;
                cout<<"\n a: ";  cin>>a;
                cout<<"\n b: ";  cin>>b;
                printf("\n 乘积（十六进制）：0x%x\n\n\n",multiple(a,b)) ;
            //  cout<<"\n乘积:"<<multiple(a,b)<<endl<<endl;

                system("pause") ;
```

```
                break;
            }
            case 3:{
                 system("cls");
                 cout<<"\n\n 平方"<<endl;
                 bitset<MAX> a;
                 cout<<"\n 请输入底数：";  cin>>a;
                 printf("\n 平方（十六进制）：0x%x\n\n",square(a)) ;
            //   cout<<"\n平方："<<square(a)<<endl;
                 system("pause") ;
                 break;
            }
            case 4:{
                system("cls");
                cout<<"\n\n 求逆"<<endl;
                bitset<MAX> a ;
                cout<<"\n 输入a：";   cin>>a;
                bitset<MAX> t = inverse(a) ;
                printf("\n 逆（十六进制）：0x%x",t) ;
            //  cout<<"\n逆："<<inverse(a)<<endl;
                printf("\n\n 与原来的乘积（十六进制）：0x%x\n\n",multiple(t,a)) ;
                system("pause") ;
                break;
            }
            case 5:{
                system("cls");
                cout<<"\n\n 学号计算\n\n"<<endl;
                studentId();
                system("pause") ;
                break;
            }
            default:
                system("cls");
                cout<<"\n\n 退出"<<endl;
                return 0;
        }
    }
}
```

5.2 加法运算

在程序中输入两个加数，初始化$bitset$类，之后两数直接异或即可。

```
void add(){
    string a,b;
    cout<<"\n 请输入两个加数："<<endl;
    cout<<"\n a: ";  cin>>a;
    cout<<"\n b: ";  cin>>b;

    bitset<MAX> x(a), y(b);
    bitset<MAX> ans = x^y ;
    printf("\n answre（十六进制）: 0x%x\n\n\n",ans);
//  cout<<"ans="<<ans<<endl;
}
```

5.3 乘法运算

首先声明两个范围都是$2*M$的变量ans，$temp$，其中$temp=a$，遍历b，如果$b[i]=1$，$temp$就左移i位，再与原来的ans异或，最后为了确保ans在有限域内，还需要$mod\ p(x)$。

```
bitset<MAX> multiple(bitset<MAX> a, bitset<MAX> b){  //完成!
    bitset<2*MAX> ans;

    bitset<2*MAX> temp(a.to_string());   //注意每次temp都是等于a

    for(int i=0;i<MAX;i++){
        if(b[i])  ans^=temp<<i;
    }
    return mod(ans);
}
```

5.4 平方运算

当a的第i位是1的话，答案的第$2i$位也会是1。为确保答案在有限域内，还需要$mod\ p(x)$。

```
bitset<MAX> square(bitset<MAX> a){   //完成!
    bitset<2*MAX> ans ;

    for(int i=0;i<MAX;i++){
        if(a[i]){
            ans[2*i] = 1 ;
        }
    }

    return mod(ans);
}
```

5.5 求逆运算

利用拓展欧几里得算法，存在$q(x)$，有$a(x)=p(x)*q(x)+r(x)$，$0\le|r(x)|<|p(x)|$

i	a(x)	q(x)	ans
0	a(x)	—	1
1	p(x)	q_1(x)	0
2	a(x)-p(x)*q1(x)	q_2(x)	1
3	$p(x)-q_2(x)[a(x)=a(x)-p(x)*q(x)]$	$q_3(x)$	$q_2(x)$
.....	.....	.....	.....
n-1	a'(x)	$q_{n-1}(x)$	ans'(x)
n	a''(x)	$q_n(x)$	ans''(x)
n+1	0 $(a'(x)=a''(x)*q_n(x))$		$ans'(x)-q_n(x)*ans''(x)$

所以，最后的答案就是当$a(x)$等于0的时候的 $ans'(x)-q_n(x)*ans''(x)$。

当$a(x)$的最高次数小于$p(x)$的时，直接交换两者的值，以及交换ans的上下两列的值。

在有限域$GF(2^{127})$中，减法就是加法，也就是异或。

```
bitset<MAX> inverse(bitset<MAX> _a){
    //贝祖定理    拓展欧几里得
    bitset<2*MAX> ans, s, a;
    bitset<2*MAX> p=bitset<2*MAX>(px.to_string()) ;
    ans[0] = 1;

    a = bitset<2*MAX>(_a.to_string());

    while(degree(a)){
        int diff = degree(a)-degree(p);
        if(diff<0){
            bitset<2*MAX> temp=a;
            a=p;
            p=temp;

            temp=s;
            s=ans;
            ans=temp;

            diff = -diff ;
        }
        a ^= p<<diff ;
        ans ^= s<<diff ;
    }

    bitset<MAX> t=bitset<MAX>(ans.to_string().substr(MAX)) ;
//  cout<<"与原来的乘积："<<multiple(t,_a)<<endl;
    return  t;

}
```

5.6 模运算模块

模运算是为了把次数大于127的多项式化成次数小于127的，所以，首先计算a的最高次数，如果小于127，直接返回原来的a即可。否则，就对次数大于127的项重新计算。

```
int degree(const bitset<2*MAX> &a){
    for(int i=2*MAX-1;i>=0;i--){
        if(a[i]==1)  return i;
    }
    return 0;
}
```

首先ans等于a中次数小于127的项的和，接着遍历次数项大于等于127的每一项，用$r(x)*x^{i-127+1}$代替，也就是rx左移$127-1$位，再与ans相加。

最后检查此时的ans，如果次数还是大于等于127，则继续像刚才一样计算。最后得出最高次数小于127的多项式（注意bitset的范围转换即可）。

```
bitset<MAX> mod(bitset<2*MAX> &a){

    bitset<2*MAX> _r = bitset<2*MAX>(rx.to_string()) ,
                  ans = bitset<2*MAX>(a.to_string().substr(MAX)) ;


    while(degree(a)>=MAX-1){
        ans.reset();
        ans = bitset<2*MAX>(a.to_string().substr(MAX+1)) ;
        for(int i=MAX;i<2*MAX;i++){
            if(a[i])   ans ^= _r<<(i-MAX) ;
        }
        a=ans;
    }

    return bitset<MAX>(ans.to_string().substr(MAX)) ;

}
```

5.7 学号计算及幂运算

利用平方乘算法，初始化ans等于1，把指数化成二进制，从高位遍历，若是1，则指数就再乘以底数。在计算平方、乘积时，每得出一个结果就要模$p(x)$，这已经在相应的函数内进行。

转成二进制：

```
void getB(int value, int array[],int &len){
    int i=0;
    while(value!=0){
        array[i++] = value % 2 ;
        value /= 2;
        len++ ;
    }
}
```

```
bitset<MAX> getExp(bitset<MAX> a){
    int exp = 20190911 , len=0;
    int array[MAX] ={0} ;
```

```
    memset(array, 0, MAX ) ;
    getB(exp,array,len) ;

    bitset<MAX> ans(1);

    int i=len-1;

    while(i>=0){
        ans = square(ans) ;
        if(array[i--])
            ans = multiple(ans, a) ;
    }

    return ans ;
}
```

a）化成有限域中的元素也就是其多项式模$p(x)$，及5.6模块。

b）求学号的逆，及5.5模块。

c）求幂运算，及5.7模块。

```
void studentId(){
    bitset<2*MAX> id;
    cout<<" 学号（十六进制）: ";
    scanf("%x",&id);
    cout<<"\n your id is "<<id<<endl;

    bitset<MAX> a = mod(id) ;
    cout<<"\n 化成有限域中的元素a: "<<a<<endl;
    printf(" (十六进制: 0x%x)\n\n",a) ;

    bitset<MAX> t = inverse(a) ;
    cout<<"\n 学号的逆: "<<t<<endl;
    printf(" (十六进制: 0x%x)\n\n",t) ;
    cout<<"与原来的乘积: "<<multiple(inverse(a),a)<<endl;

    cout<<"\n a^20190911: "<<getExp(a)<<endl ;
    printf(" (十六进制: 0x%x)\n\n\n",getExp(a)) ;
}
```

六、程序实现及分析

6.1 首先进入main函数模块

```
请选择你要进行的计算:
1-加法    2-乘法    3-平方    4-求逆    5-学号计算    6-退出
choice:
```

6.2 选择加法运算

以下的结果都会化成十六进制输出，经过手动计算，实验结果正确。

```
加法
请输入两个加数：
a：10101101000000001010100101010
b：101010100000101
answre（十六进制）：0x15a0402f

请按任意键继续. . .
```

6.3 选择乘法运算

经验算，结果正确。

```
乘法
请输入两个乘数：
a：10101011111111111111111111101010011010100100001000010000001001
b：10101010
乘积（十六进制）：0xfbf7adfa

请按任意键继续. . .
```

6.4 选择平方运算

经计算，结果正确。

```
平方
请输入底数：10110011101011110010101011010101010100101
平方（十六进制）：0x14444411
请按任意键继续. . .
```

6.5 选择求逆运算

得到a的逆之后，再与a相乘，得到乘积1，答案正确。

```
求逆
输入a：11101010101011111111101010100111100011111000111001111110010101
逆（十六进制）：0x6187d17
与原来的乘积（十六进制）：0x1
请按任意键继续. . .
```

6.7 选择学号计算

经过验算，结果正确。

```
 学号计算


 学号（十六进制）：17341125

 your id is 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000010111001101000001000100100101

 化成有限域中的元素a：000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000010111001101000001000100100101
 (十六进制：0x17341125)


 学号的逆：11000011101011111110101000001000010100101101101100100000011111010001000000111111100001010
1001000000001011000101111011100101111000
 (十六进制：0x58bdcb8)

与原来的乘积：00000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000001

 a^20190911：0101000010100000100110001110111001110100010111111001010011000001001000100100110110111100
11000010010101011010100110111010000110011
 (十六进制：0xab537433)


请按任意键继续. . .
```

七、性能体现

加法	乘法	平方	求逆	学号运算
0.001723 s	0.005617 s	0.004184 s	0.005086 s	0.4437 s

电脑配置：

八、实验总结

- 有限域的相关知识之前在信息安全数学基础这门课中学了一部分，但是学的不是很到位，所以还是上网学习了一下，看看大佬们是怎么用大佬代码来构造一个有限域。学习了以后，感觉这个有限域的构造主要还是体现在运算当中，每一次计算得到的多项式的最高次数都不能超过127，其中的不可约多项式就起到了很大的作用，这个多项式模$p(x)$之后，自然就是有限域的元素了。

- 各种运算首先是正常进行，我在纸上多写几个式子就发现了其规律，当然，信息安全数学中的代数计算也发挥了极大的作用，类比于整数的计算，可以得到多项式的计算其实也是在整数计算上拓展的，但是万变不离其宗嘛。这样一来就可以得到每一种计算的方法了。其中，我发现如果把$p(x)$的范围定为MAX就不能得到正确的逆，MAX+1才可以，但是不知道为什么……
- 这次试验也让我感受到了从信息安全数学基础到密码学的深度上的递进，在百度学习大佬的代码的时候，也看到了AES的计算的博客，这也许就是应用吧，把所学的运用到实际的密码上，也真的是有趣，怀着敬畏之心期待接下来的实验。最后谢谢老师和TA啦~