**欧几里得算法：**

首先考虑一下：

对于任意两个正整数 a,b ，都有：

a=kb+r (k,r∈N)1

所以有：

r=a%b （在这里，%指的是取余运算）1

然后我们假设 c 是 a 和 b 的最大公约数，即

c=gcd(a,b)1

然后，我们就能得到

c|a c|b （x|y 表示 x 能够整除 y ， y能被x整除 , 也就是y/x是整数）1

然后又因为上面那个式子，有：

r=a−kb1

所以有：

c|r1

那么我们就可以知道，既然a和b的因数也是b和（a%b）的因数，那么它们的最大公因数肯定也是相同的。

整合一下上面的式子，我们可以得到：

c=gcd(b,r)1

即

gcd(a,b)=gcd(b,a%b) ----------gcd(a,b)表示a和b的最大公约数1

而且

gcd(a,0) = a1

辗转相除法函数代码：

int gcd(int a,int b)//就是欧几里得算法函数，即辗转相除法，求gcd(a,b)

{

int c;

while(b!=0)

{

c=a;

a=b;

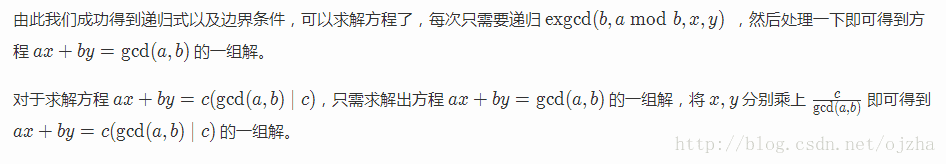
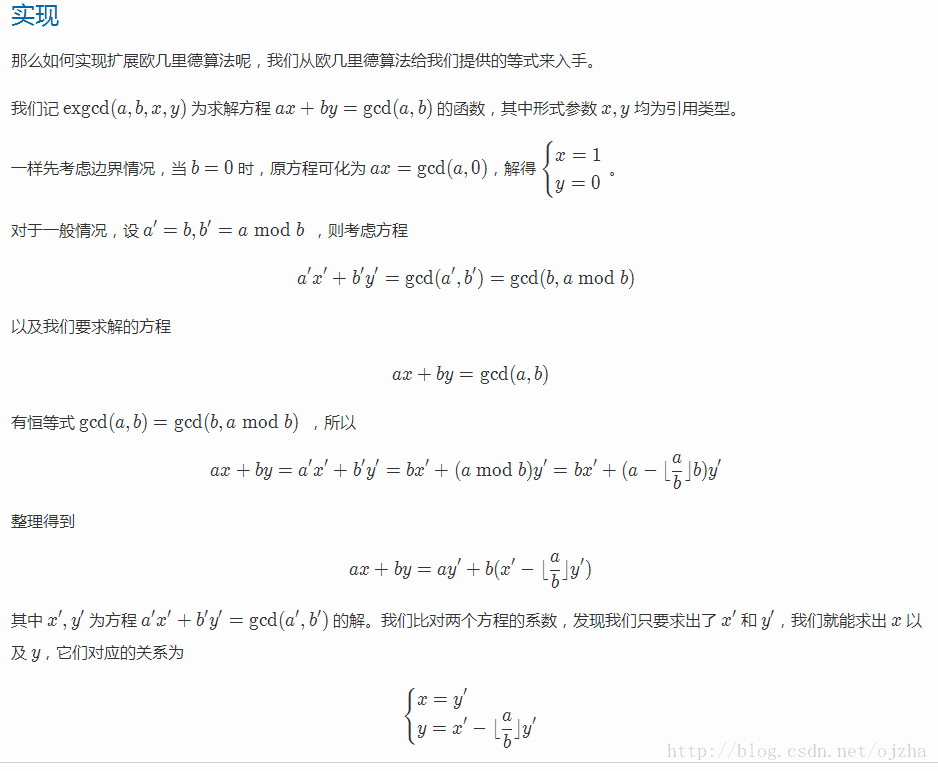
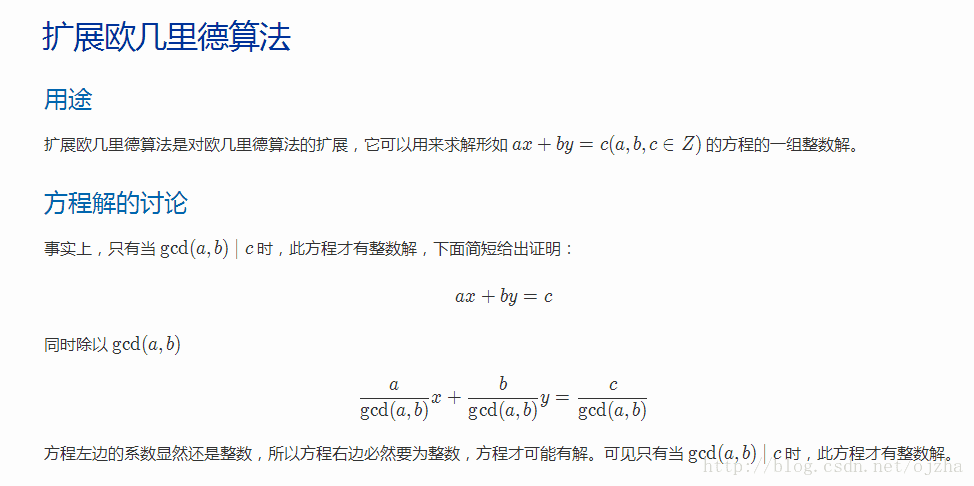
b=c%b;

}

return a;

}

扩展欧几里得算法:



题目就是求关于 x 的同余方程 ax ≡ 1 (mod b)的最小正整数解

可能乍一看，ax ≡ 1 (mod b)跟上面的ax+by=gcd(a,b)这一个方程不太一样啊，没事，让我们来推导一下。

.

.

首先,题目保证了b是素数，即gcd(a,b)一定是1

我们设r=a\*x%b, 有a\*x=b\*k+r

然后ax ≡ 1 (mod b)就转换为了a\*x-b\*k=1

然后我们再设 y=-k,方程就转换成了 a\*x+b\*y=1

即a\*x + b\*y = gcd(a,b) = 1

就是妥妥的扩展欧几里得算法嘛！！！ 递归求x的值就好啦！！！！

这里我还要提一下：我们递归求出来的x可能并不是最小正整数，还看是负数，我们这时候就需要处理一下。需要将x mod p,然后加上p（为了搞定负数）,再mod p,代码就是：x = (x%p+p) % p;

#include<iostream>

#include<cstdio>

#include<cstring>

using namespace std;

long long x,y,n;//最好定全局变量

void exgcd(long long a,long long b)

{

if(b==0) //当b=0时就是遇到了特解，可以递归回去算答案了

{

x=1,y=0;

return ;

}

exgcd(b,a%b);

long long k;

k=x;

x=y;

y=k-(a/b)\*y;

}

int main()

{

long long a,p;

scanf("%d%lld",&a,&p);

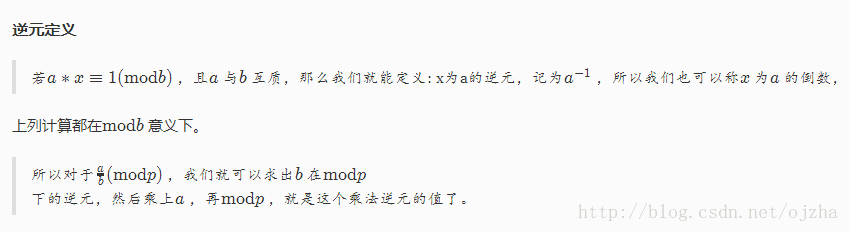
exgcd(a,p);

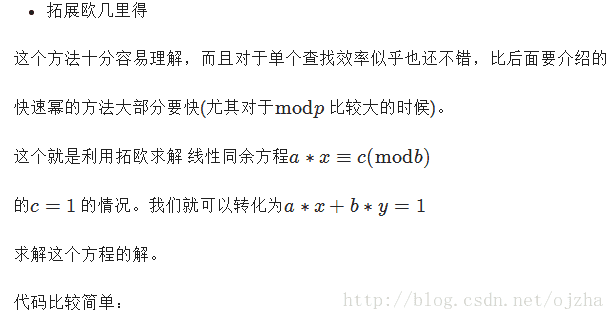
cout<<(x%p+p)%p;

return 0;

}

乘法逆元：





题目就是求关于 x 的同余方程 ax ≡ 1 (mod b)的最小正整数解

可能乍一看，ax ≡ 1 (mod b)跟上面的ax+by=gcd(a,b)这一个方程不太一样啊，没事，让我们来推导一下。

.

.

首先,题目保证了b是素数，即gcd(a,b)一定是1

我们设r=a\*x%b, 有a\*x=b\*k+r

然后ax ≡ 1 (mod b)就转换为了a\*x-b\*k=1

然后我们再设 y=-k,方程就转换成了 a\*x+b\*y=1

即a\*x + b\*y = gcd(a,b) = 1

就是妥妥的扩展欧几里得算法嘛！！！ 递归求x的值就好啦！！！！

这里我还要提一下：我们递归求出来的x可能并不是最小正整数，还看是负数，我们这时候就需要处理一下。需要将x mod p,然后加上p（为了搞定负数）,再mod p,代码就是：x = (x%p+p) % p;

#include<iostream>

#include<cstdio>

#include<cstring>

using namespace std;

long long x,y,n;//最好定全局变量

void exgcd(long long a,long long b)

{

if(b==0) //当b=0时就是遇到了特解，可以递归回去算答案了

{

x=1,y=0;

return ;

}

exgcd(b,a%b);

long long k;

k=x;

x=y;

y=k-(a/b)\*y;

}

int main()

{

long long a,p;

scanf("%d%lld",&a,&p);

exgcd(a,p);

cout<<(x%p+p)%p;

return 0;

}

中国剩余定理：

  首先，我们假设n1是满足除以3余2的一个数，比如2，5，8等等，也就是满足3\*k+2（k>=0）的一个任意数。同样，我们假设n2是满足除以5余3的一个数，n3是满足除以7余2的一个数。

     有了前面的假设，我们先从n1这个角度出发，已知n1满足除以3余2，能不能使得 n1+n2 的和仍然满足除以3余2？进而使得n1+n2+n3的和仍然满足除以3余2？

     这就牵涉到一个最基本数学定理，**如果有a%b=c,则有(a+kb)%b=c(k为非零整数)，换句话说，如果一个除法运算的余数为c，那么被除数与k倍的除数相加（或相减）的和（差）再与除数相除，余数不变。**这个是很好证明的。

     以此定理为依据，如果n2是3的倍数，n1+n2就依然满足除以3余2。同理，如果n3也是3的倍数，那么n1+n2+n3的和就满足除以3余2。这是从n1的角度考虑的，再从n2，n3的角度出发，我们可推导出以下三点：

1. 为使n1+n2+n3的和满足除以3余2，n2和n3必须是3的倍数。
2. 为使n1+n2+n3的和满足除以5余3，n1和n3必须是5的倍数。
3. 为使n1+n2+n3的和满足除以7余2，n1和n2必须是7的倍数。

    因此，为使n1+n2+n3的和作为“孙子问题”的一个最终解，需满足：

1. **n1除以3余2，且是5和7的公倍数。**
2. **n2除以5余3，且是3和7的公倍数。**
3. **n3除以7余2，且是3和5的公倍数。**

    所以，孙子问题解法的本质是从5和7的公倍数中找一个除以3余2的数n1，从3和7的公倍数中找一个除以5余3的数n2，从3和5的公倍数中找一个除以7余2的数n3，再将三个数相加得到解。在求n1，n2，n3时又用了一个小技巧，以n1为例，并非从5和7的公倍数中直接找一个除以3余2的数，而是先找一个除以3余1的数，再乘以2。

    这里又有一个数学公式，如果a%b=c，那么（a\*k）%b=a%b+a%b+…+a%b=c+c+…+c=kc（k>0）,也就是说，如果一个除法的余数为c，那么被除数的k倍与除数相除的余数为kc。展开式中已证明。

    最后，我们还要清楚一点，n1+n2+n3只是问题的一个解，并不是最小的解。如何得到最小解？我们只需要从中最大限度的减掉3，5，7的公倍数105即可。道理就是前面讲过的定理“如果a%b=c,则有(a-kb)%b=c”。所以（n1+n2+n3）%105就是最终的最小解。

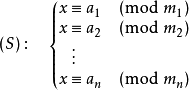
经过分析发现，中国剩余定理的孙子解法并没有什么高深的技巧，就是以下两个基本数学定理的灵活运用：

1. 如果 a%b=c , 则有 (a+kb)%b=c (k为非零整数)。
2. 如果 a%b=c，那么 (a\*k)%b=kc (k为大于零的整数)。

[回到顶部](https://www.cnblogs.com/wkfvawl/p/9633188.html#_labelTop)

**数学分析**

用现代数学的语言来说明的话，中国剩余定理给出了以下的一元线性同余方程组：



有解的判定条件，并用构造法给出了在有解情况下解的具体形式。

中国剩余定理说明：假设整数m1,m2, ... ,mn两两互质，则对任意的整数：a1,a2, ... ,an，方程组https://gss1.bdstatic.com/9vo3dSag_xI4khGkpoWK1HF6hhy/baike/s%3D20/sign=b30da9ccf1d3572c62e29bdc8b1383a3/9a504fc2d56285355c012c0493ef76c6a7ef6336.jpg有解，并且通解可以用如下方式构造得到：

设 https://gss0.bdstatic.com/94o3dSag_xI4khGkpoWK1HF6hhy/baike/s%3D223/sign=afd164d67a310a55c024d9f684444387/7af40ad162d9f2d30fcbdacaaaec8a136327cc39.jpg是整数m1,m2, ... ,mn的乘积，并设https://gss3.bdstatic.com/7Po3dSag_xI4khGkpoWK1HF6hhy/baike/s%3D203/sign=56bcfe3fc55c1038207ec9c28110931c/91ef76c6a7efce1b22fa36efac51f3deb58f65c6.jpg是除了mi以外的n- 1个整数的乘积。

设

https://gss3.bdstatic.com/-Po3dSag_xI4khGkpoWK1HF6hhy/baike/s%3D59/sign=98cf0fe8ff039245a5b5e10687947d4e/562c11dfa9ec8a133826aec5f403918fa0ecc0d3.jpg为

https://gss2.bdstatic.com/-fo3dSag_xI4khGkpoWK1HF6hhy/baike/s%3D20/sign=3421af3261d0f703e2b292dc08fa9d75/91ef76c6a7efce1b232431efac51f3deb48f658c.jpg模

https://gss2.bdstatic.com/9fo3dSag_xI4khGkpoWK1HF6hhy/baike/s%3D17/sign=9757be9e4410b912bbc1f2f9c3fd6213/f3d3572c11dfa9ecd055af3261d0f703918fc198.jpg的数论倒数(

https://gss0.bdstatic.com/-4o3dSag_xI4khGkpoWK1HF6hhy/baike/s%3D14/sign=600e2894abcc7cd9fe2d30dd3801d19f/dcc451da81cb39db7a5c542fd8160924ab18302e.jpg为

https://gss2.bdstatic.com/-fo3dSag_xI4khGkpoWK1HF6hhy/baike/s%3D20/sign=987bd370be003af349badb60342ace9c/d0c8a786c9177f3ec9a1cf7478cf3bc79f3d5635.jpg模

https://gss2.bdstatic.com/-fo3dSag_xI4khGkpoWK1HF6hhy/baike/s%3D17/sign=41ce7c820246f21fcd345a54f624e601/dc54564e9258d10925da9bcbd958ccbf6c814d89.jpg意义下的逆元)

https://gss1.bdstatic.com/9vo3dSag_xI4khGkpoWK1HF6hhy/baike/s%3D270/sign=1e2b1996a1ec8a13101a50e7c7039157/5ab5c9ea15ce36d39c3ebe4932f33a87e950b194.jpg

方程组

https://gss1.bdstatic.com/9vo3dSag_xI4khGkpoWK1HF6hhy/baike/s%3D20/sign=b30da9ccf1d3572c62e29bdc8b1383a3/9a504fc2d56285355c012c0493ef76c6a7ef6336.jpg的通解形式为

https://gss3.bdstatic.com/-Po3dSag_xI4khGkpoWK1HF6hhy/baike/s%3D486/sign=cb3ee938d72a283447a637036db5c92e/2fdda3cc7cd98d10aa8ef514223fb80e7bec90b9.jpg

在模

https://gss1.bdstatic.com/-vo3dSag_xI4khGkpoWK1HF6hhy/baike/s%3D16/sign=e44c04e9c0cec3fd8f3ea373d6887f24/a8ec8a13632762d0aa5d5d9aa3ec08fa513dc6b0.jpg的意义下，方程组

https://gss1.bdstatic.com/9vo3dSag_xI4khGkpoWK1HF6hhy/baike/s%3D20/sign=b30da9ccf1d3572c62e29bdc8b1383a3/9a504fc2d56285355c012c0493ef76c6a7ef6336.jpg只有一个解：

https://gss0.bdstatic.com/94o3dSag_xI4khGkpoWK1HF6hhy/baike/s%3D162/sign=6e46f98494510fb37c197391eb32c893/8ad4b31c8701a18b0f5da572942f07082838fe5d.jpg

[复制代码](javascript:void(0);)

1 void exgcd(int a1,int b,int &x,int &y)

2 {

3 if(b==0)

4 {

5 x=1;

6 y=0;

7 return ;

8 }

9 exgcd(b,a1%b,x,y);

10 int t=x;

11 x=y;

12 y=t-(a1/b)\*y;

13 }

14 int CRT(int a[],int m[],int n)

15 {

16 int M=1,ans=0,t,x,y;

17 for(int i=0; i<n; i++)

18 {

19 M\*=m[i];///M为除数乘积

20 }

21 for(int i=0; i<n; i++)

22 {

23 t=M/m[i];///除了mi以外的n-1个整数乘积

24 exgcd(t,m[i],x,y);///求逆元，https://gss1.bdstatic.com/9vo3dSag_xI4khGkpoWK1HF6hhy/baike/s%3D270/sign=1e2b1996a1ec8a13101a50e7c7039157/5ab5c9ea15ce36d39c3ebe4932f33a87e950b194.jpg由扩展欧几里得转换成t\*ti+m[i]\*y=1来求ti

25 ans=(ans+a[i]\*x\*t)%M;

26 }

27 return (ans+M)%M;

28 }

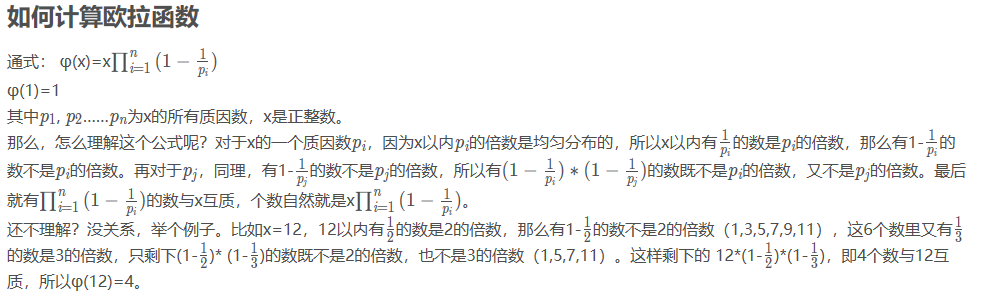
[复制代码](javascript:void(0);)

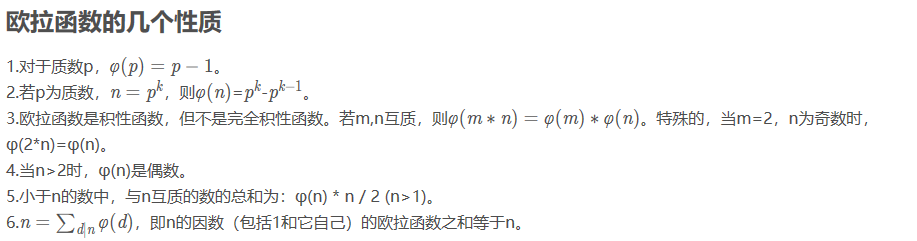
一个正整数K，给出K Mod 一些质数的结果，求符合条件的最小的K。例如，K % 2 = 1, K % 3 = 2, K % 5 = 3。符合条件的最小的K = 23。

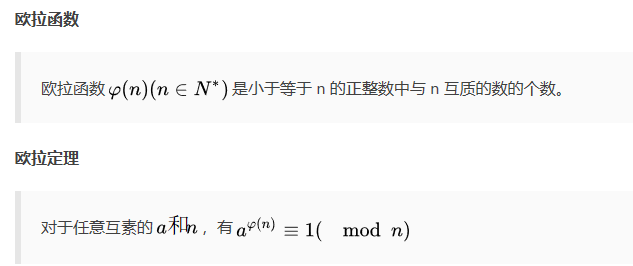
欧拉函数：φ(n)表示从1~n-1中有多少个数与n互素。 φ(1) = 1

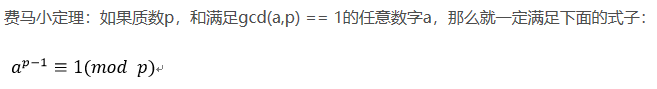
积性函数：

先介绍一下什么是积性函数，后面将会用到。若当m与n互质时，f(m∗n)=f(m)∗f(n)f(m∗n)=f(m)∗f(n) f(m\*n)=f(m)\*f(n)f(m∗n)=f(m)∗f(n)，那么f是积性函数。若对任意正整数，都有f(m\*n)=f(m)\*f(n)成立，则f是完全积性函数。









费马小定理是欧拉函数的性质与欧拉定理的特殊情况。

拉格朗日定理：若H是G的一个子群，则#H|#G,即：#H能够整除#G