# 乘法逆元

求普通逆元（即a和m互素）请参见1061（extend\_gcd）

Poj 1061

扩展欧几里得算法

扩展欧几里德算法是欧几里得算法的扩展。

基本算法：对于不完全为 0 的非负整数 a，b，gcd（a，b）表示 a，b 的最大公约数，必然存在整数对 x，y ，使得 gcd（a，b）=ax+by。

01 证明：设 a>b。

02

03 　　推理1，显然当 b=0，gcd（a，b）=a。此时 x=1，y=0；//推理1

04

05 　　推理2，ab!=0 时

06

07 　　设 ax1+by1=gcd(a,b);

08

09 　　bx2+(a mod b)y2=gcd(b,a mod b);

10

11 　　根据朴素的欧几里德原理有 gcd(a,b)=gcd(b,a mod b);

12

13 　　则:ax1+by1=bx2+(a mod b)y2;

14

15 　　即:ax1+by1=bx2+(a-(a/b)\*b)y2=ay2+bx2-(a/b)\*by2;

16

17 　　根据恒等定理得：x1=y2; y1=x2-(a/b)\*y2;//推理2

18

19 这样我们就得到了求解 x1,y1 的方法：x1，y1 的值基于 x2，y2.

20

21 　 上面的思想是以递归定义的，因为 gcd 不断的递归求解一定会有个时候 b=0，所以递归可以结束。

扩展欧几里德的递归代码：

01 #include <iostream>

02 using namespace std;

03

04 int exgcd(int a,int b,int & x,int & y){

05 if(b == 0){

06 //根据上面的推理1，基本情况

07 x = 1;

08 y = 0;

09 return a;

10 }

11 int r = exgcd(b, a%b, x, y);

12 //根据推理2

13 int t = y;

14 y = x - (a/b)\*y;

15 x = t;

16 return r;

17 }

18

19

扩展欧几里德算法的应用

（1）求解不定方程

用扩展欧几里得算法解不定方程ax+by=c;

这个应该比较好理解了，两个可以同乘以k

1 bool linear\_equation(int a,int b,int c,int &x,int &y)

2 {

3 int d=exgcd(a,b,x,y);

4 if(c%d)

5 return false;

6 int k=c/d;

7 x\*=k; y\*=k; //求得的只是其中一组解

8 return true;

9 }

（2）求解模线性方程（线性同余方程）

同余方程 ax≡b (mod n) (也就是 ax % n = b) 对于未知数 x 有解，当且仅当 gcd(a,n) | b (也就是 b % (gcd(a,n))==0 )。且方程有解时，方程有 gcd(a,n) 个解。

求解方程 ax≡b (mod n) 相当于求解方程 ax+ ny= b, (x, y为整数)

1 在方程 3x ≡ 2 (mod 6) 中， d = gcd(3,6) = 3 ，3 不整除 2，因此方程无解。

2

3 在方程 5x ≡ 2 (mod 6) 中， d = gcd(5,6) = 1，1 整除 2，因此方程在{0,1,2,3,4,5} 中恰有一个解: x=4。

证明略去，直接说算法：

首先看一个简单的例子：

5x=4(mod3)

解得x = 2,5,8,11,14…….

由此可以发现一个规律，就是解的间隔是3.

那么这个解的间隔是怎么决定的呢？

如果可以设法找到第一个解，并且求出解之间的间隔，那么就可以求出模的线性方程的解集了.

我们设解之间的间隔为dx.

那么有

a\*x = b(mod n);

a\*(x+dx) = b(mod n);

两式相减，得到:

a\*dx(mod n)= 0;

也就是说a\*dx就是a的倍数，同时也是n的倍数，即a\*dx是a 和 n的公倍数.为了求出dx,我们应该求出a 和 n的最小公倍数,此时对应的dx是最小的.

设a 和 n的最大公约数为d,那么a 和 n 的最小公倍数为(a\*n)/d.

即a\*dx = a\*n/d;

所以dx = n/d. (d = gcd(a,n) )

因此解之间的间隔就求出来了.

（3）求解模的逆元；

同余方程ax≡b (mod n)，如果 gcd(a,n)== 1，则方程只有唯一解。

在这种情况下，如果 b== 1，同余方程就是 ax=1 (mod n ),gcd(a,n)= 1。

这时称求出的 x 为 a 的对模 n 乘法的逆元。

对于同余方程 ax= 1(mod n )， gcd(a,n)= 1 的求解就是求解方程

ax+ ny= 1，x, y 为整数。这个可用扩展欧几里德算法求出，原同余方程的唯一解就是用扩展欧几里德算法得出的 x 。

对于正整数http://img.blog.csdn.net/20140613102654328和http://img.blog.csdn.net/20140613102712781，如果有http://img.blog.csdn.net/20140613102734984，那么把这个同余方程中http://img.blog.csdn.net/20140613102856531的最小正整数解叫做http://img.blog.csdn.net/20140613102654328模http://img.blog.csdn.net/20140613102712781的逆元。

逆元一般用扩展欧几里得[**算法**](http://lib.csdn.net/base/datastructure)来求得，如果http://img.blog.csdn.net/20140613102712781为素数，那么还可以根据费马小定理得到逆元为http://img.blog.csdn.net/20140613103413828。

推导过程如下

http://img.blog.csdn.net/20140613104248984

 现在来看一个逆元最常见问题，求如下表达式**（已知http://img.blog.csdn.net/20140613104752312即a%b=0）**

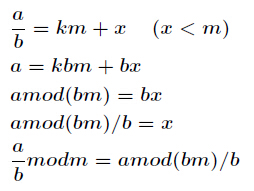
http://img.blog.csdn.net/20140613104619203

 当然这个经典的问题有很多方法，最常见的就是扩展欧几里得，如果http://img.blog.csdn.net/20140613102712781是素数，还可以用费马小定理。

但是你会发现费马小定理和扩展欧几里得算法求逆元是有局限性的，它们都会要求http://img.blog.csdn.net/20140613102654328与http://img.blog.csdn.net/20140613102712781互素。实际上我们还有一种通用的求逆元方法，适合所有情况。公式如下（即要求a%b==0）

http://img.blog.csdn.net/20140613105646406

现在我们来证明它，已知http://img.blog.csdn.net/20140613104752312，证明步骤如下

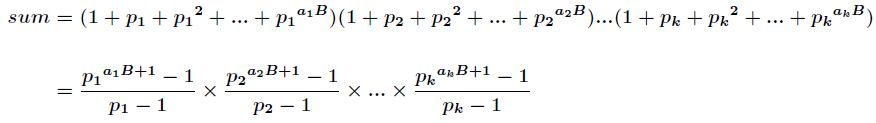


接下来来实战一下，看几个关于逆元的题目。

**题目：**<http://poj.org/problem?id=1845>

**题意：**给定两个正整数http://img.blog.csdn.net/20140613112543656和http://img.blog.csdn.net/20140613112553093，求http://img.blog.csdn.net/20140613112622265的所有因子和对**9901**取余后的值。

**分析：**很容易知道，先把http://img.blog.csdn.net/20140613112543656分解得到http://img.blog.csdn.net/20140613113059203，那么得到http://img.blog.csdn.net/20140613113255234，那么http://img.blog.csdn.net/20140613112622265的所有因子和的表达式如下



因为(p^n)-1一定可以凑出p-1与另一项的乘积 故可以用上式转化mod数

注意本题转化后的mod数自较大 故可以用二分乘法

long long multi(long long at,long long bt,long long mod){

long long ans = 0;

at%=mod;

while(bt){

if(bt & 1){

ans = (ans + at) % mod;

bt--;

}

bt >>= 1;

at= (at + at) % mod;

}

return ans;

}

其实有些题需要用到http://img.blog.csdn.net/20140613122746828模http://img.blog.csdn.net/20140613122821000的所有逆元（http://img.blog.csdn.net/20140613122821000为奇质数）。那么如果用快速幂求时间复杂度为http://img.blog.csdn.net/20140613123226296，

如果对于一个**1000000**级别的素数http://img.blog.csdn.net/20140613122821000，这样做的时间复杂度是很高了。实际上有http://img.blog.csdn.net/20140613123603656的算法，有一个递推式如下

http://img.blog.csdn.net/20140613123955765

它的推导过程如下，设http://img.blog.csdn.net/20140613124932968，那么

http://img.blog.csdn.net/20140613125025984

对上式两边同时除http://img.blog.csdn.net/20140613125121234，进一步得到

http://img.blog.csdn.net/20140613125208000

再把http://img.blog.csdn.net/20140613125307921和http://img.blog.csdn.net/20140613125324765替换掉，最终得到

http://img.blog.csdn.net/20140613123955765

初始化http://img.blog.csdn.net/20140613125520578，这样就可以通过递推法求出http://img.blog.csdn.net/20140613122746828模奇素数http://img.blog.csdn.net/20140613122821000的所有逆元了。

另外http://img.blog.csdn.net/20140613122746828模http://img.blog.csdn.net/20140613122821000的所有逆元值对应http://img.blog.csdn.net/20140613122746828中所有的数，比如http://img.blog.csdn.net/20140613125913078，那么http://img.blog.csdn.net/20140613130144843对应的逆元是http://img.blog.csdn.net/20140613130219921。