# ****大步小步****

**Poj 2417**

**【Baby\_Step,Gaint\_Step定义】**

高次同余方程。   BL == N (mod P)

求解最小的L。由于数据范围很大，暴力不行

这里用到baby\_step,giant\_step算法。意为先小步，后大步。

令L=i\*m+j  (m=ceil(sqrt(p-1)))，

那么原式化为 B^(i\*m)\*B^j==N(MOD P)

继而化为B^j===N\*B^(-i\*m)(MOD P)

我们先预处理B^0,B^1,B^2……B^(m-1),存入HASH表。,这一步就是baby-step,每次移动1

然后求出B^-m,枚举i，如果存在B^(-i\*m)存在于HASH表中，说明存在解L=i\*m+j    ，这一步为giant\_step,每次移动m

至于B^(-m)的求法，可以先求出B的逆元，也就是B^-1。

注意以上解法是最基本的，只能对于gcd(B,P)==1

【解体思路】

我们可以做一个等价  
x = i \* m + j  ( 0 <= i < m, 0 <=j < m) m = Ceil ( sqrt( C) )  
而这么分解的目的无非是为了转化为:  
(A^i)^m \* A^j = B ( mod C)  
之后做少许暴力的工作就可以解决问题：  
(1) for i = 0 -> m, 插入Hash (i, A^i mod C)  
(2) 枚举 i ,对于每一个枚举到的i,令 AA = (A^m)^i mod C  
我们有**AA \* A^j = B (mod C)**  
显然AA,B,C均已知，而由于C为素数,那么(AA,C)无条件为1  
于是对于这个模方程解的个数唯一(可以利用扩展欧几里得或 欧拉定理来求解)  
那么对于得到的唯一解X,在Hash表中寻找,如果找到，则返回 **i \* m + j**  
注意:**由于i从小到大的枚举,而Hash表中存在的j必然是对于某个剩余系内的元素X 是最小的(就是指标)**  
**所以显然此时就可以得到最小解**  
如果需要得到 x > 0的解，那么只需要在上面的步骤中判断 当 i \* m + j > 0 的时候才返回  
 上面算法总的时间复杂度接近O(sqrt(C)\*log(C)) (C是模）

<http://poj.org/problem?id=3243>

题意：

和上一题一样，还是求A^x = B( mod C )的最小x值，但是这题和上题有个不同点就是

这题的C没有限制条件，也就是说这题并没有规定C必须是质数。

思路：

     还是用 babystep\_gaintstep**[算法](http://lib.csdn.net/base/datastructure" \o "算法与数据结构知识库" \t "_blank)**求解。但是这题并不能用POJ\_2417的算法，直接套该算法，下面简要说明一下不能用的原因。首先我们有必要归纳一下用babystep算法解题

的步骤： (1) 、求M = ceil( sqrt(C) ) ；(2) for(i=0;i<M;i++)   hash( i , A^i ) ；(3) 求D = A^M

%C；(4) 、r = 1 ; for( i = 0 ; i < M ; i++ )  ex\_gcd(r , C , x , y ) ; res = x \* B % C  ; jj = find( res)

如果找到了这时候的jj，则答案就是i\*M+jj，如果没有找到，则res = res \* D % C，继续循

环查找，如果最终都没有找到，则输出无解。 在上述的步骤中，如果题目中没有告诉我们

gcd(A , C) = 1，则我们上述的方法是错误的，原因就在于第4步，求res的时候。因为如果

我们无法保证gcd(A , C) = 1 ，也就不能保证gcd(r ,C) = 1（因为D=A^M, r = D^i），所有在

用 扩展欧几里得求出r\*x + C\*y = gcd(r,C ) 的一个解x0之后，原方程:r\*x+C\*y = B的解

x = x0 \* B / gcd(r,C) + i\*C / gcd(r,C) ，但是我们这个时候并不能计算出gcd(r,C)，因为此时

的r本来就是经过取余之后得出的，并不能直接用来求gcd，因此我们上述的普通babystep

算法就会出错了。

      这样我们就要换一种处理的方法了，这里介绍一种AC大牛博客上的一种“消因子”的方法，

具体内容请看这里：[AC大牛](http://hi.baidu.com/aekdycoin/blog/item/b317ca18bb24334942a9ad55.html" \t "_blank)。经过上面的分析我们很清楚接下去的处理应该从哪方面着手，就是应该从不能求出gcd(r , C)入手。一种思想就是既然无法求， 那我每次只要保证gcd(r, C) = 1那样就可以想普通babystep一样求解了，既然要保证gcd(r,C) =1 ，而 r = (A^M)^i,因此归根到底还是要求gcd( A , C )  = 1。每次我们 都消去A,C的一个因子，然后对B,C, D进行如下的处理：B/=tmp;C/=tmp ;

D = D\* A/tmp%C ，

这样经过b轮的消因子之后，gcd(A,C) = 1， 接下去我们就可以用普通

的babystep求解出方程：A^x = B'( mod C' ) 的解 res1， 原方程的解就是 res = res1 + b。

下面给出这种方法正确的简要证明；一开始我们要求的方程是：A^x = B( mod C )，也就是求一个最小的x，使得A^x + C\*y = B，通过消因子， 我们不断在方程两遍消去gcd(A,C)，这样方程就可以变成 D\*A^x1 + C'\*y1 = B'，很简单就可以证明上式中 x = x1 + b ; y = y1 的（只要在方程的两边分别将消去的因子乘回去等式还是保持不变的）。这样我们的问题就转化为了求x1和y1，即D\*A^x1 = B'( mod C' )，此时gcd( A , C') = 1，这样我们就可以用普通的babystep求出上述式子的解x1，同时也就求出了x，这样本题就解决了 。

详细证明，一开始的方程等价于A^x\*a+C\*b=B(a,b是整数)，现在gcd(A，C)!=1,不妨令t=gcd(A,C)

我们考虑方程(A/t)A^x'%(C/t)=(B/t)的解x'与原来的解x有什么关系。

显然现在的方程等价于(A/t)A^x'\*a'+(C/t)\*b'=B/t.

两端均乘以t得到：A^(x'+1)\*a'+C\*b'=B

由系数相等有:x=x'+1,a=a',b=b'.

那么我们就有一种方法算出x了。先算出x',再加上1就行了。

考虑如何求出x',现在的方程如果A,C依旧不互质，就继续迭代将系数除以最大公约数，同时前面的常数增大。

然后求出解之后再加上总共除的次数就好了。

       但是上述的方法还是有一个bug的，也就是说，我们用babystep求出的x1>=0，所以上述的

方法只能求出x >= b的解，这样我们自然就会想到如果有一个解x < b怎么办，上述方法就会出

先错误了，因此我们这里还需要改进。考虑b的最大值是多少，考虑每次我们消去的因子数都最小也就是2，这样我们就可以得到b的最大值就是log(C)，这样我们只要保证每次log(C)之内的

解都特判一下， 就不会出现我们刚才的问题了， 所以我们要在进行上述处理之前进行一次for循环 ，特判0 - log(C)直接的x是否能成为解，接下去再用上述的“消因子”算法。

     最后不得不佩服发明这种算法的人的神奇，将O(C)复杂度的判断，分两级判断将复杂度降低

到O( sqrt(C) )，所以就是为什么叫" babystep\_gaintstep "了， 哈哈。