[逆元详解](http://blog.csdn.net/acdreamers/article/details/8220787)

对于正整数和，如果有，那么把这个同余方程中的最小正整数解叫做模的逆元。

逆元一般用扩展欧几里得算法来求得，如果为素数，那么还可以根据费马小定理得到逆元为。 推导过程如下

求现在来看一个逆元最常见问题，求如下表达式的值**（已知）**



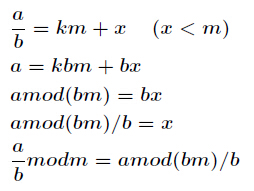
当然这个经典的问题有很多方法，最常见的就是扩展欧几里得，如果是素数，还可以用费马小定理。

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，它们都会要求与互素。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



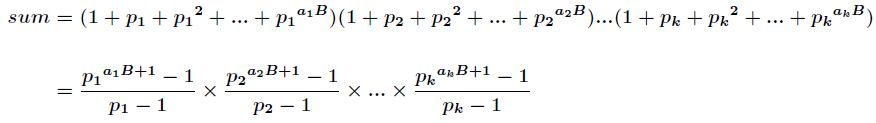
现在我们来证明它，已知，证明步骤如下



接下来来实战一下，看几个关于逆元的题目。

**题意：**给定两个正整数和，求的所有因子和对**9901**取余后的值。

**分析：**很容易知道，先把分解得到，那么得到，那么 的所有因子和的表达式如下



所以我们有两种做法。第一种做法是二分求等比数列之和。

**代码：**

1. **bool** prime[N];
2. **int** p[N];
3. **int** cnt;
5. **void** isprime()
6. {
7. cnt = 0;
8. memset(prime,**true**,**sizeof**(prime));
9. **for**(**int** i=2; i<N; i++)
10. {
11. **if**(prime[i])
12. {
13. p[cnt++] = i;
14. **for**(**int** j=i+i; j<N; j+=i)
15. prime[j] = **false**;
16. }
17. }
18. }
20. LL power(LL a,LL b)
21. {
22. LL ans = 1;
23. a %= MOD;
24. **while**(b)
25. {
26. **if**(b & 1)
27. {
28. ans = ans \* a % MOD;
29. b--;
30. }
31. b >>= 1;
32. a = a \* a % MOD;
33. }
34. **return** ans;
35. }
37. LL sum(LL a,LL n)
38. {
39. **if**(n == 0) **return** 1;
40. LL t = sum(a,(n-1)/2);
41. **if**(n & 1)
42. {
43. LL cur = power(a,(n+1)/2);
44. t = (t + t % MOD \* cur % MOD) % MOD;
45. }
46. **else**
47. {
48. LL cur = power(a,(n+1)/2);
49. t = (t + t % MOD \* cur % MOD) % MOD;
50. t = (t + power(a,n)) % MOD;
51. }
52. **return** t;
53. }
55. **void** Solve(LL A,LL B)
56. {
57. LL ans = 1;
58. **for**(**int** i=0; p[i]\*p[i] <= A; i++)
59. {
60. **if**(A % p[i] == 0)
61. {
62. **int** num = 0;
63. **while**(A % p[i] == 0)
64. {
65. num++;
66. A /= p[i];
67. }
68. ans \*= sum(p[i],num\*B) % MOD;
69. ans %= MOD;
70. }
71. }
72. **if**(A > 1)
73. {
74. ans \*= sum(A,B) % MOD;
75. ans %= MOD;
76. }
77. cout<<ans<<endl;
78. }
80. **int** main()
81. {
82. LL A,B;
83. isprime();
84. **while**(cin>>A>>B)
85. Solve(A,B);
86. **return** 0;
87. }

第二种方法就是用等比数列求和公式，但是要用逆元。用如下公式即可



因为可能会很大，超过**int**范围，所以在快速幂时要二分乘法。

**代码：**

1. **const** **int** N = 10005;
2. **const** **int** MOD = 9901;
4. **bool** prime[N];
5. **int** p[N];
6. **int** cnt;
8. **void** isprime()
9. {
10. cnt = 0;
11. memset(prime,**true**,**sizeof**(prime));
12. **for**(**int** i=2; i<N; i++)
13. {
14. **if**(prime[i])
15. {
16. p[cnt++] = i;
17. **for**(**int** j=i+i; j<N; j+=i)
18. prime[j] = **false**;
19. }
20. }
21. }
23. LL multi(LL a,LL b,LL m)
24. {
25. LL ans = 0;
26. a %= m;
27. **while**(b)
28. {
29. **if**(b & 1)
30. {
31. ans = (ans + a) % m;
32. b--;
33. }
34. b >>= 1;
35. a = (a + a) % m;
36. }
37. **return** ans;
38. }
40. LL quick\_mod(LL a,LL b,LL m)
41. {
42. LL ans = 1;
43. a %= m;
44. **while**(b)
45. {
46. **if**(b & 1)
47. {
48. ans = multi(ans,a,m);
49. b--;
50. }
51. b >>= 1;
52. a = multi(a,a,m);
53. }
54. **return** ans;
55. }
57. **void** Solve(LL A,LL B)
58. {
59. LL ans = 1;
60. **for**(**int** i=0; p[i]\*p[i] <= A; i++)
61. {
62. **if**(A % p[i] == 0)
63. {
64. **int** num = 0;
65. **while**(A % p[i] == 0)
66. {
67. num++;
68. A /= p[i];
69. }
70. LL M = (p[i] - 1) \* MOD;
71. ans \*= (quick\_mod(p[i],num\*B+1,M) + M - 1) / (p[i] - 1);
72. ans %= MOD;
73. }
74. }
75. **if**(A > 1)
76. {
77. LL M = MOD \* (A - 1);
78. ans \*= (quick\_mod(A,B+1,M) + M - 1) / (A - 1);
79. ans %= MOD;
80. }
81. cout<<ans<<endl;
82. }
84. **int** main()
85. {
86. LL A,B;
87. isprime();
88. **while**(cin>>A>>B)
89. Solve(A,B);
90. **return** 0;
91. }

其实有些题需要用到模的所有逆元，这里为奇质数。那么如果用快速幂求时间复杂度为，

如果对于一个**1000000**级别的素数，这样做的时间复杂度是很高了。实际上有的算法，有一个递推式如下



它的推导过程如下，设，那么



对上式两边同时除，进一步得到



再把和替换掉，最终得到



初始化，这样就可以通过递推法求出模奇素数的所有逆元了。

另外模的所有逆元值对应中所有的数，比如，那么对应的逆元是。

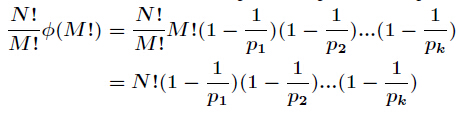
**题意：**求中互质的数的个数，其中。

**分析：**因为，所以，我们很容易知道如下结论

**对于两个正整数和，如果是的倍数，那么中与互素的数的个数为**

本结论是很好证明的，因为中与互素的个数为，又知道，所以

     结论成立。那么对于本题，答案就是



      其中为小于等于的所有素数，先筛选出来即可。由于最终答案对一个质数取模，所以要用逆元，这里

      求逆元就有技巧了，用刚刚介绍的递推法预处理，否则会**TLE**的。

**代码：**

1. **typedef** **long** **long** LL;
2. **const** **int** N = 10000005;
4. bitset<N> prime;
6. **void** isprime()
7. {
8. prime.set();
9. **for**(**int** i=2; i<N; i++)
10. {
11. **if**(prime[i])
12. {
13. **for**(**int** j=i+i; j<N; j+=i)
14. prime[j] = **false**;
15. }
16. }
17. }
19. LL ans1[N],ans2[N];
20. LL inv[N];
22. **int** main()
23. {
24. isprime();
25. **int** MOD,m,n,T;
26. scanf("%d%d",&T,&MOD);
27. ans1[0] = 1;
28. **for**(**int** i=1; i<N; i++)
29. ans1[i] = ans1[i-1] \* i % MOD;
30. inv[1] = 1;
31. **for**(**int** i=2;i<N;i++)
32. {
33. **if**(i >= MOD) **break**;
34. inv[i] = (MOD - MOD / i) \* inv[MOD % i] % MOD;
35. }
36. ans2[1] = 1;
37. **for**(**int** i=2; i<N; i++)
38. {
39. **if**(prime[i])
40. {
41. ans2[i] = ans2[i-1] \* (i - 1) % MOD;
42. ans2[i] = ans2[i] \* inv[i % MOD] % MOD;
43. }
44. **else**
45. {
46. ans2[i] = ans2[i-1];
47. }
48. }
49. **while**(T--)
50. {
51. scanf("%d%d",&n,&m);
52. LL ans = ans1[n] \* ans2[m] % MOD;
53. printf("%lld\n",ans);
54. }
55. **return** 0;
56. }

接下来还有一个关于逆元的有意思的题目，描述如下



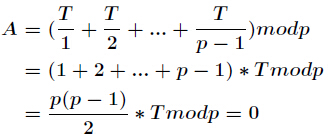
**证明：**由



     其中

     所以只需要证明，而我们知道模的逆元对应全部

中的所有数，既是单射也是满射。 所以进一步得到



      证明完毕。