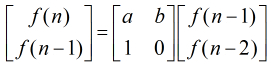
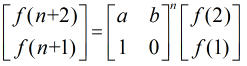
[广义Fibonacci数列找循环节](http://blog.csdn.net/acdreamers/article/details/25616461)

**问题：**给定，满足，求的循环节长度。

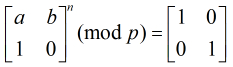
**分析：**我们知道矩阵的递推关系如下



 然后继续有



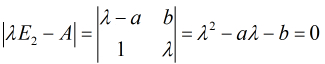
     那么，现在的问题就转化为求最小的，使得



     所以我们可以先找出符合条件的一个，然后枚举它的因子，找最小的。设



     为了好解决问题，我们需要对矩阵进行相似对角化，即，我们先来求的特征值。



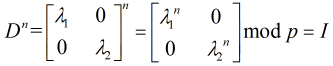
     解得的特征值为



     也就是说的相似对角矩阵为



      因为我们知道，所以当时，， 由于



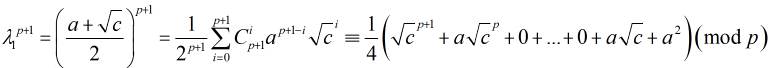
      继续得到



      设，那么分情况讨论：

       （1）是模的二次剩余，由费马小定理得时，

       （2）是模的二次非剩余，则有



           根据欧拉准则有



           那么继续得到



           然后由费马小定理有，同理有

           所以，当时，

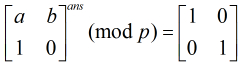
       （3）时，由于不存在，所以无法完成相似对角化，好在这种情况不存在。

      所以综上所述：

是模的二次剩余时，枚举的因子

是模的二次非剩余时，枚举的因子

      找最小的因子，使得



      成立。

1. #include <iostream>
2. #include <string.h>
3. #include <algorithm>
4. #include <stdio.h>
5. #include <math.h>
7. **using** **namespace** std;
8. **typedef** **long** **long** LL;
9. **const** **int** N = 2;
10. **const** LL MOD = 1000000007;
12. LL fac[2][505];
13. **int** cnt,ct;
15. LL pri[6] = {2, 3, 7, 109, 167, 500000003};
16. LL num[6] = {4, 2, 1, 2, 1, 1};
18. **struct** Matrix
19. {
20. LL m[N][N];
21. } ;
23. Matrix A;
24. Matrix I = {1, 0, 0, 1};
26. Matrix multi(Matrix a,Matrix b)
27. {
28. Matrix c;
29. **for**(**int** i=0; i<N; i++)
30. {
31. **for**(**int** j=0; j<N; j++)
32. {
33. c.m[i][j]  =0;
34. **for**(**int** k=0; k<N; k++)
35. {
36. c.m[i][j] += a.m[i][k] \* b.m[k][j];
37. c.m[i][j] %= MOD;
38. }
39. }
40. }
41. **return** c;
42. }
44. Matrix power(Matrix A,LL n)
45. {
46. Matrix ans = I, p = A;
47. **while**(n)
48. {
49. **if**(n & 1)
50. {
51. ans = multi(ans,p);
52. n--;
53. }
54. n >>= 1;
55. p = multi(p,p);
56. }
57. **return** ans;
58. }
60. LL quick\_mod(LL a,LL b)
61. {
62. LL ans = 1;
63. a %= MOD;
64. **while**(b)
65. {
66. **if**(b & 1)
67. {
68. ans = ans \* a % MOD;
69. b--;
70. }
71. b >>= 1;
72. a = a \* a % MOD;
73. }
74. **return** ans;
75. }
77. LL Legendre(LL a,LL p)
78. {
79. LL t = quick\_mod(a,(p-1)>>1);
80. **if**(t == 1) **return** 1;
81. **return** -1;
82. }
84. **void** dfs(**int** dept,LL product = 1)
85. {
86. **if**(dept == cnt)
87. {
88. fac[1][ct++] = product;
89. **return**;
90. }
91. **for**(**int** i=0; i<=num[dept]; i++)
92. {
93. dfs(dept+1,product);
94. product \*= pri[dept];
95. }
96. }
98. **bool** OK(Matrix A,LL n)
99. {
100. Matrix ans = power(A,n);
101. **return** ans.m[0][0] == 1 && ans.m[0][1] == 0 &&
102. ans.m[1][0] == 0 && ans.m[1][1] == 1;
103. }
105. **int** main()
106. {
107. fac[0][0] = 1;
108. fac[0][1] = 2;
109. fac[0][2] = 500000003;
110. fac[0][3] = 1000000006;
111. LL a,b,c,d;
112. **while**(cin>>a>>b>>c>>d)
113. {
114. LL t = a \* a + 4 \* b;
115. A.m[0][0] = a;
116. A.m[0][1] = b;
117. A.m[1][0] = 1;
118. A.m[1][1] = 0;
119. **if**(Legendre(t,MOD) == 1)
120. {
121. **for**(**int** i=0; i<4; i++)
122. {
123. **if**(OK(A,fac[0][i]))
124. {
125. cout<<fac[0][i]<<endl;
126. **break**;
127. }
128. }
129. }
130. **else**
131. {
132. ct = 0;
133. cnt = 6;
134. dfs(0,1);
135. sort(fac[1],fac[1]+ct);
136. **for**(**int** i=0;i<ct;i++)
137. {
138. **if**(OK(A,fac[1][i]))
139. {
140. cout<<fac[1][i]<<endl;
141. **break**;
142. }
143. }
144. }
145. }
146. **return** 0;

}