数论

1. 逆元详解 ……………………………………… 2
2. 数论中的若干推论 …………………………… 11
3. 欧拉函数和欧拉定理 ………………………… 13
4. 广义斐波那契数列 …………………………… 15
5. 伯努利数 …………………………………… 22
6. 表为平方和 ………………………………… 27
7. Bell数 ……………………………………… 40
8. 五边形数定理 ………………………………… 43
9. Catalan数 …………………………………… 47
10. 数学中的若干定理和数列前n项 ………… 52
11. [Fibonacci数列的幂和](http://blog.csdn.net/acdreamers/article/details/23039571) …………………… 54
12. Fib数模n循环节 …………………………… 58

[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）**



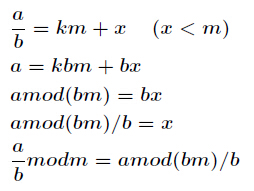
当然这个经典的问题有很多方法，最常见的就是扩展欧几里得，如果是素数，还可以用费马小定理。

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



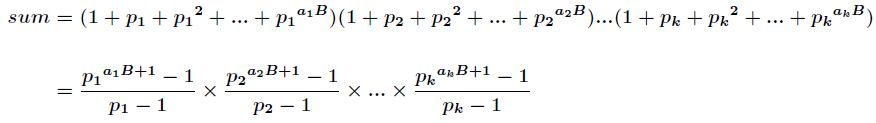
现在我们来证明它，已知，证明步骤如下



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;
4. **void** isprime()
5. {
6. cnt = 0;
7. memset(prime,**true**,**sizeof**(prime));
8. **for**(**int** i=2; i<N; i++)
9. {
10. **if**(prime[i])
11. {
12. p[cnt++] = i;
13. **for**(**int** j=i+i; j<N; j+=i)
14. prime[j] = **false**;
15. }
16. }
17. }
18. LL power(LL a,LL b)
19. {
20. LL ans = 1;
21. a %= MOD;
22. **while**(b)
23. {
24. **if**(b & 1)
25. {
26. ans = ans \* a % MOD;
27. b--;
28. }
29. b >>= 1;
30. a = a \* a % MOD;
31. }
32. **return** ans;
33. }
34. LL sum(LL a,LL n)
35. {
36. **if**(n == 0) **return** 1;
37. LL t = sum(a,(n-1)/2);
38. **if**(n & 1)
39. {
40. LL cur = power(a,(n+1)/2);
41. t = (t + t % MOD \* cur % MOD) % MOD;
42. }
43. **else**
44. {
45. LL cur = power(a,(n+1)/2);
46. t = (t + t % MOD \* cur % MOD) % MOD;
47. t = (t + power(a,n)) % MOD;
48. }
49. **return** t;
50. }
52. **void** Solve(LL A,LL B)
53. {
54. LL ans = 1;
55. **for**(**int** i=0; p[i]\*p[i] <= A; i++)
56. {
57. **if**(A % p[i] == 0)
58. {
59. **int** num = 0;
60. **while**(A % p[i] == 0)
61. {
62. num++;
63. A /= p[i];
64. }
65. ans \*= sum(p[i],num\*B) % MOD;
66. ans %= MOD;
67. }
68. }
69. **if**(A > 1)
70. {
71. ans \*= sum(A,B) % MOD;
72. ans %= MOD;
73. }
74. cout<<ans<<endl;
75. }
76. **int** main()
77. {
78. LL A,B;
79. isprime();
80. **while**(cin>>A>>B)
81. Solve(A,B);
82. **return** 0;
83. }

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。

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. }
39. LL quick\_mod(LL a,LL b,LL m)
40. {
41. LL ans = 1;
42. a %= m;
43. **while**(b)
44. {
45. **if**(b & 1)
46. {
47. ans = multi(ans,a,m);
48. b--;
49. }
50. b >>= 1;
51. a = multi(a,a,m);
52. }
53. **return** ans;
54. }
56. **void** Solve(LL A,LL B)
57. {
58. LL ans = 1;
59. **for**(**int** i=0; p[i]\*p[i] <= A; i++)
60. {
61. **if**(A % p[i] == 0)
62. {
63. **int** num = 0;
64. **while**(A % p[i] == 0)
65. {
66. num++;
67. A /= p[i];
68. }
69. LL M = (p[i] - 1) \* MOD;
70. ans \*= (quick\_mod(p[i],num\*B+1,M) + M - 1) / (p[i] - 1);
71. ans %= MOD;
72. }
73. }
74. **if**(A > 1)
75. {
76. LL M = MOD \* (A - 1);
77. ans \*= (quick\_mod(A,B+1,M) + M - 1) / (A - 1);
78. ans %= MOD;
79. }
80. cout<<ans<<endl;
81. }
83. **int** main()
84. {
85. LL A,B;
86. isprime();
87. **while**(cin>>A>>B)
88. Solve(A,B);
89. **return** 0;
90. }

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。

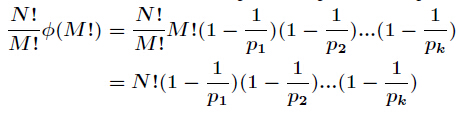
**题意：**求中互质的数的个数，其中。

**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。

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. }
18. LL ans1[N],ans2[N];
19. LL inv[N];
21. **int** main()
22. {
23. isprime();
24. **int** MOD,m,n,T;
25. scanf("%d%d",&T,&MOD);
26. ans1[0] = 1;
27. **for**(**int** i=1; i<N; i++)
28. ans1[i] = ans1[i-1] \* i % MOD;
29. inv[1] = 1;
30. **for**(**int** i=2;i<N;i++)
31. {
32. **if**(i >= MOD) **break**;
33. inv[i] = (MOD - MOD / i) \* inv[MOD % i] % MOD;
34. }
35. ans2[1] = 1;
36. **for**(**int** i=2; i<N; i++)
37. {
38. **if**(prime[i])
39. {
40. ans2[i] = ans2[i-1] \* (i - 1) % MOD;
41. ans2[i] = ans2[i] \* inv[i % MOD] % MOD;
42. }
43. **else**
44. {
45. ans2[i] = ans2[i-1];
46. }
47. }
48. **while**(T--)
49. {
50. scanf("%d%d",&n,&m);
51. LL ans = ans1[n] \* ans2[m] % MOD;
52. printf("%lld\n",ans);
53. }
54. **return** 0;
55. }       

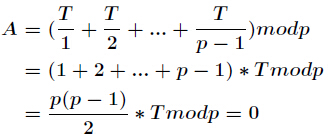
**证明：**由



     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。

[数论中的若干定理](http://blog.csdn.net/acdreamers/article/details/7909480)

**素数定理：**记为小于等于的素数个数，那么有

**定理：**设，，那么有

**定理：**设，，那么

**定理：**设，那么的值为

**（1）**为素数，那么答案就是

**（2）**有多个素因子，那么答案就是

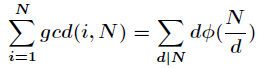
**（3）**只有一个素因子，那么答案就是该素因子

**定理：**设为**Fib**数，那么有

合的数为，不能组合的数的个数为



**定理：**

****

**定理：**

**定理：**任何个连续的正整数的乘积均可被整除

关于上述定理的两个结论

**（1）**如果是素数，那么均能被整除

**证明：**如果，那么有，由于与素数，那么有

，所以对所有的有



         很明显能被整除。

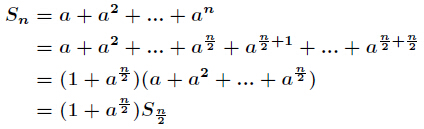
**（2）**如果是素数，那么有

**证明：**由结论**（1）**很容易得到，一般性的结论可以重复此结论而得到。

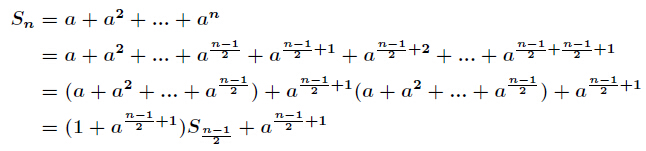
 有效地求表达式的值。

**（1）**当时，

**（2）**当时，那么有



**（3）**当时，那么有



欧拉函数与欧拉定理

先来介绍几个与欧拉函数有关的定理：

定理一：设m与n是互素的正整数，那么

定理二：当n为奇数时，有。

因为2n是偶数，偶数与偶数一定不互素，所以只考虑2n与小于它的奇数互素的情况，则恰好就等于n的欧拉函数值。

定理三：设p是素数，a是一个正整数，那么

关于这个定理的证明用到容斥：

由于表示小于与互素数的正整数个数，所以用减去与它不互素的数的个数就行了。

那么小于与不互素数的个数就是p的倍数个数，有个。所以定理得证。

定理四：设为正整数n的素数幂分解，那么



这个定理可以根据定理一和定理三证明，其实用到的就是容斥。如果对容斥熟悉，其实完全就可以直接容斥。

定理五：设n是一个正整数，那么



这个其实可以看莫比乌斯反演就明白了。

定理六：设m是正整数，(a,m)=1，则：是同于方程的解。

定理七：如果n大于2，那么n的欧拉函数值是偶数。

求欧拉函数值：

1. **int** phi(**int** n)
2. {
3. **int** i,rea=n;
4. **for**(i=2;i\*i<=n;i++)
5. {
6. **if**(n%i==0)
7. {
8. rea=rea-rea/i;
9. **while**(n%i==0)  n/=i;
10. }
11. }
12. **if**(n>1)
13. rea=rea-rea/n;
14. **return** rea;
15. }

利用递推法求欧拉函数值：

算法原理：开始令i的欧拉函数值等于它本身，如果i为偶数，可以利用定理二变为求奇数的。

若p是一个正整数满足，那么p是素数，在遍历过程中如果遇到欧拉函数值等于自身的情况，那么

说明该数为素数。把这个数的欧拉函数值改变，同时也把能被该素因子整除的数改变。

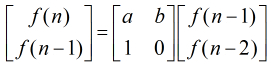
1. **void** phi()
2. {
3. **for**(**int** i=1; i<N; i++)  p[i] = i;
4. **for**(**int** i=2; i<N; i+=2) p[i] >>= 1;
5. **for**(**int** i=3; i<N; i+=2)
6. {
7. **if**(p[i] == i)
8. {
9. **for**(**int** j=i; j<N; j+=i)
10. p[j] = p[j] - p[j] / i;
11. }
12. }

}

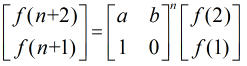
[广义Fibonacci数列找循环节](http://blog.csdn.net/acdreamers/article/details/25616461)

**问题：**给定，满足，求的循环节长度。

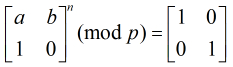
**分析：**我们知道矩阵的递推关系如下



 然后继续有



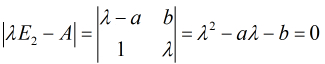
     那么，现在的问题就转化为求最小的，使得



     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。



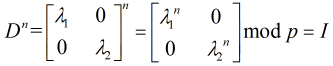
     解得的特征值为



     也就是说的相似对角矩阵为



      因为我们知道，所以当时，， 由于



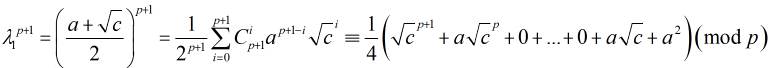
      继续得到



      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，

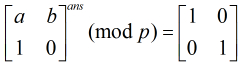
       （3）时，由于不存在，所以无法完成相似对角化，好在这种情况不存在。

      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;

}

[伯努利数与自然数幂和](http://blog.csdn.net/acdreamers/article/details/38929067)

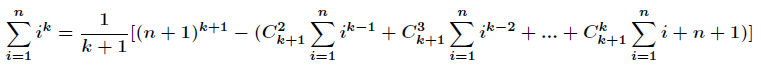
**分析：**其实求自然数的幂和方法有很多种，先来看看普通的递推求法，由于



     那么对于所有的累加得到



  进一步得到

  可以看出这是一个递推式，如果我们记



     那么得到如下递归式



     递归出口是



     为了提高效率，在递归的时候需要记忆化。由于要用到大数，用**Java**实现。 **代码：**

**import** java.math.\*;

1. **import** java.util.\*;
3. **public** **class** Main {
5. **public** **static** **final** **int** N = 105;
6. **public** **static** **final** BigInteger FLAG = (BigInteger.ZERO).subtract(BigInteger.ONE);
7. **public** **static** BigInteger[][] C = **new** BigInteger[N][N];
8. **public** **static** BigInteger[] ans = **new** BigInteger[N];
10. **public** **static** **void** Init(){
11. **for**(**int** i=0; i<N; i++){
12. C[i][0] = C[i][i] = BigInteger.ONE;
13. **if**(i == 0) **continue**;
14. **for**(**int** j=1; j<i; j++)
15. C[i][j] = C[i-1][j].add(C[i-1][j-1]);
16. }
17. }
19. **public** **static** BigInteger Solve(BigInteger n, **int** k){
20. **if**(ans[k].compareTo(FLAG) != 0){
21. **return** ans[k];
22. }
23. **if**(k == 1){
24. ans[k] = ((n.add(BigInteger.ONE)).multiply(n)).divide(BigInteger.valueOf(2));
25. **return** ans[k];
26. }
27. BigInteger tmp = BigInteger.ONE;
28. **for**(**int** i=0; i<k+1; i++){
29. tmp = tmp.multiply(n.add(BigInteger.ONE));
30. }
31. tmp = tmp.subtract(n.add(BigInteger.ONE));
32. BigInteger sum = BigInteger.ZERO;
33. **for**(**int** i=1; i<k; i++){
34. BigInteger t = C[k+1][i+1].multiply(Solve(n, k-i));
35. sum = sum.add(t);
36. }
37. ans[k] = (tmp.subtract(sum)).divide(BigInteger.valueOf(k+1));
38. **return** ans[k];
39. }
41. **public** **static** **void** main(String[] args){
42. Init();
43. Scanner cin = **new** Scanner(System.in);
44. **while**(cin.hasNext()){
45. BigInteger n = cin.nextBigInteger();
46. **int** k = cin.nextInt();
47. **for**(**int** i=0; i<N; i++){
48. ans[i] = FLAG;
49. }
50. System.out.println(Solve(n, k));
51. }
52. }
53. }

**分析：**本题题意就是求**自然数的幂和**，但是它的**case**比较多。对于求幂和本身就需要的时间复杂度，如果继

     续用上述方法来求自然数的幂和，**5000**个**case**会**TLE**，接下来介绍另一个求自然数幂和的方法，它是基于伯努利数的，公式描述如下



     可以看出只要我们预处理出每一项，就可以在线性时间内求得自然数的幂和。前面的倒数可以用递推法求逆元

     预处理，组合数也可以预处理，也可以先预处理，现在关键是如何预处理**伯努利数**。  **伯努利数**满足条件，且有



     那么继续得到

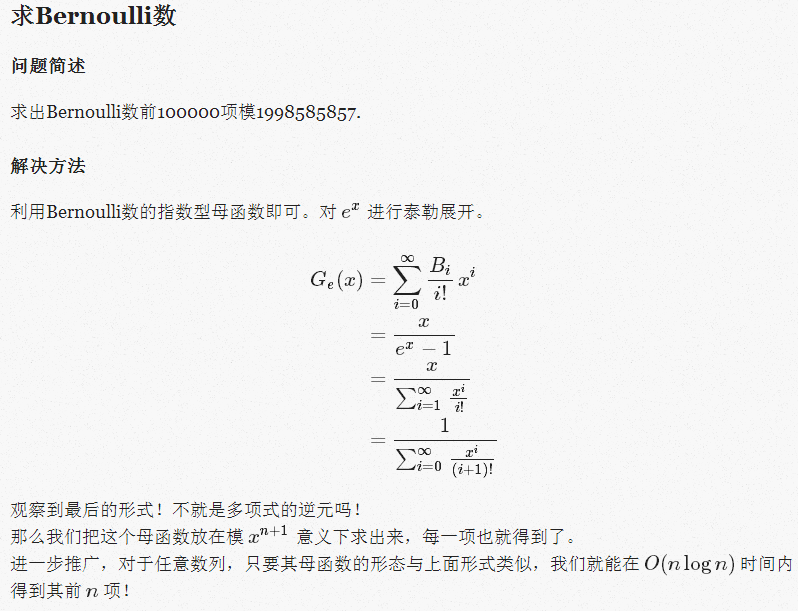


     这就是伯努利数的递推式，逆元部分同样可以预处理。

**代码：**

1. LL C[N][N];
2. LL B[N],Inv[N];
3. LL Tmp[N];
4. LL n;
6. **void** Init()
7. {
8. //预处理组合数
9. **for**(**int** i=0; i<N; i++)
10. {
11. C[i][0] = C[i][i] = 1;
12. **if**(i == 0) **continue**;
13. **for**(**int** j=1; j<i; j++)
14. C[i][j] = (C[i-1][j] % MOD + C[i-1][j-1] % MOD) % MOD;
15. }
16. //预处理逆元
17. Inv[1] = 1;
18. **for**(**int** i=2; i<N; i++)
19. Inv[i] = (MOD - MOD / i) \* Inv[MOD % i] % MOD;
20. //预处理伯努利数
21. B[0] = 1;
22. **for**(**int** i=1; i<N; i++)
23. {
24. LL ans = 0;
25. **if**(i == N - 1) **break**;
26. **for**(**int** j=0; j<i; j++)
27. {
28. ans += C[i+1][j] \* B[j];
29. ans %= MOD;
30. }
31. ans \*= -Inv[i+1];
32. ans = (ans % MOD + MOD) % MOD;
33. B[i] = ans;
34. }
35. }
37. LL Work(**int** k)
38. {
39. LL ans = Inv[k+1];
40. LL sum = 0;
41. **for**(**int** i=1; i<=k+1; i++)
42. {
43. sum += C[k+1][i] \* Tmp[i] % MOD \* B[k+1-i] % MOD;
44. sum %= MOD;
45. }
46. ans \*= sum;
47. ans %= MOD;
48. **return** ans;
49. }
51. **int** main()
52. {
53. **int** T;
54. Init();
55. scanf("%d", &T);
56. **while**(T--)
57. {
58. **int** k;
59. scanf("%I64d %d", &n, &k);
60. n %= MOD;
61. Tmp[0] = 1;
62. **for**(**int** i=1; i<N; i++)
63. Tmp[i] = Tmp[i-1] \* (n + 1) % MOD;
64. printf("%I64d\n", Work(k));
65. }
66. **return** 0;
67. }

**分析：**本题与上题不同的是值比较大，达到**50000**，如果采用同样的方法，会**TLE**的。那么必定要进行优化。



**表为平方和问题**

简单来说就是对于丢番图方程来说，求满足条件的整数解。当然，根据的范围不同，所使用的方法也可能会不一样，接下来将会以几个题为例进行深度剖析。

**题意：**给定一个圆的方程，其中有，求在这个圆上有多少个整数点。

**分析：**这个问题比较简单，因为对于丢番图方程来说，整数解的个数可以通过如下方法计算



     其中，满足

**（1）**当为偶数时，有

**（2）**当为奇数时，有

     那么，本题的方法就很明确了，先对进行大数分解，然后搜出所有因子，再枚举所有因子判断累加即可。由于比较简单，省略代码。

**题意：**给定一个正整数，其中，求该正整数最少能够使用多少个正整数的平方和来表示。

**分析：**先来认识几个重要的定理，如下

**（1）费马平方和定理**

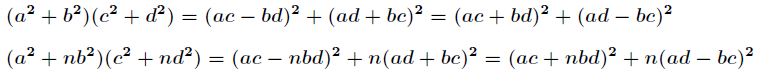
       奇质数能表示为两个平方数之和的充分必要条件是该素数被**4**除余**1**

**（2）费马平方和定理的拓展定理**

       正整数能表示为两平方数之和的充要条件是在它的标准分解式中，形如素因子的指数是偶数

**（3）Brahmagupta–Fibonacci identity**

如果两个整数都能表示为两个平方数之和，则它们的积也能表示为两个平方数之和。公式及拓展公式为



        从这个定理可以看出：如果不能表示为三个数的平方和，那么也就不能表示为两个数的平方和。

**（4）四平方和定理**

每个正整数都可以表示成四个整数的平方数之和

**（5）表为3个数的平方和条件**

正整数能表示为三个数的平方和的充要条件是不能表示成的形式，其中和为非负整数。

      通过上面的几个重要的定理，就可以来做这题了。

1. **int** Work(LL n)
2. {
3. **while**(n % 4 == 0) n >>= 2;
4. **if**(n % 8 == 7) **return** 4;
5. LL i = 8, t = 9;
6. **while**(t <= n)
7. {
8. **while**(n % t == 0) n /= t;
9. i += 8;
10. t += i;
11. }
12. **if**(n == 1) **return** 1;
13. **if**(n % 2 == 0) n >>= 1;
14. **if**(n % 4 == 3) **return** 3;
15. LL k = 3;
16. **while**(k \* k <= n)
17. {
18. **if**(n % k == 0) **return** 3;
19. k += 4;
20. }
21. **return** 2;
22. }
24. **int** main()
25. {
26. LL n;
27. **while**(cin>>n)
28. cout<<Work(n)<<endl;
29. **return** 0;
30. }

上面第**13**行至第**19**行的作用是消去中所有素数的偶次幂，这里为，每次增加**2**，而每次相邻两项满

足，所以每次增加**8**。

 接下来，开始进入我们今天的重点，求丢番图方程的解。这里很大，范围是。

**分析：**通过**Brahmagupta–Fibonacci identity**知道，如果两个整数都能表示为两个平方数之和，则它们的积也

     能表示为两个平方数之和，即



      那么，我们可以先对素因子分解，采用**Pollard-rho**分解法，然后对于每个素数，求出的

     解，然后进行合并即可。而求解丢番图方程（其中为素数）的方法如下

**（1）**首先判断奇素数是否满足条件，如果是才有解，并且解唯一，否则无解。

**（2）**找出二次同余方程的一个解

**（3）**对奇素数和进行欧几里德辗转相除运算，记录每次的余数为（注意第一次的余数为），当满

         足时结束，此时的就是丢番图方程中的，进而通过得到。

1. **const** **int** Times = 10;
2. **const** **int** N = 65;
4. **using** **namespace** std;
5. **typedef** **long** **long** LL;
7. LL ct, cnt;
8. LL fac[N], num[N];
10. **struct** T
11. {
12. LL p, d;
13. };
14. LL w;
16. **struct** Pair
17. {
18. LL x, y;
19. **bool** operator < (**const** Pair &a) **const**
20. {
21. **if**(a.x == x) **return** a.y > y;
22. **return** a.x > x;
23. }
24. };
25. Pair node[N];
26. set<Pair> Set;
27. set<Pair> Mem;
29. **int** sign[2][2] = {{-1, 1}, {1, -1}};
31. LL gcd(LL a, LL b)
32. {
33. **return** b? gcd(b, a % b) : a;
34. }
36. LL multi(LL a, LL b, LL m)
37. {
38. LL ans = 0;
39. a %= m;
40. **while**(b)
41. {
42. **if**(b & 1)
43. {
44. ans = (ans + a) % m;
45. b--;
46. }
47. b >>= 1;
48. a = (a + a) % m;
49. }
50. **return** ans;
51. }
53. LL quick\_mod(LL a, LL b, LL m)
54. {
55. LL ans = 1;
56. a %= m;
57. **while**(b)
58. {
59. **if**(b & 1)
60. {
61. ans = multi(ans, a, m);
62. b--;
63. }
64. b >>= 1;
65. a = multi(a, a, m);
66. }
67. **return** ans;
68. }
70. **bool** Miller\_Rabin(LL n)
71. {
72. **if**(n == 2) **return** **true**;
73. **if**(n < 2 || !(n & 1)) **return** **false**;
74. LL m = n - 1;
75. **int** k = 0;
76. **while**((m & 1) == 0)
77. {
78. k++;
79. m >>= 1;
80. }
81. **for**(**int** i=0; i<Times; i++)
82. {
83. LL a = rand() % (n - 1) + 1;
84. LL x = quick\_mod(a, m, n);
85. LL y = 0;
86. **for**(**int** j=0; j<k; j++)
87. {
88. y = multi(x, x, n);
89. **if**(y == 1 && x != 1 && x != n - 1) **return** **false**;
90. x = y;
91. }
92. **if**(y != 1) **return** **false**;
93. }
94. **return** **true**;
95. }
97. LL pollard\_rho(LL n, LL c)
98. {
99. LL i = 1, k = 2;
100. LL x = rand() % (n - 1) + 1;
101. LL y = x;
102. **while**(**true**)
103. {
104. i++;
105. x = (multi(x, x, n) + c) % n;
106. LL d = gcd((y - x + n) % n, n);
107. **if**(1 < d && d < n) **return** d;
108. **if**(y == x) **return** n;
109. **if**(i == k)
110. {
111. y = x;
112. k <<= 1;
113. }
114. }
115. }
117. **void** find(LL n, **int** c)
118. {
119. **if**(n == 1) **return**;
120. **if**(Miller\_Rabin(n))
121. {
122. fac[ct++] = n;
123. **return** ;
124. }
125. LL p = n;
126. LL k = c;
127. **while**(p >= n) p = pollard\_rho(p, c--);
128. find(p, k);
129. find(n / p, k);
130. }
132. **void** Partion(LL n)
133. {
134. ct = 0;
135. find(n, 120);
136. sort(fac, fac + ct);
137. num[0] = 1;
138. **int** k = 1;
139. **for**(**int** i=1; i<ct; i++)
140. {
141. **if**(fac[i] == fac[i-1])
142. ++num[k-1];
143. **else**
144. {
145. num[k] = 1;
146. fac[k++] = fac[i];
147. }
148. }
149. cnt = k;
150. }
152. **bool** Filter()
153. {
154. **for**(**int** i = 0; i < cnt; i++)
155. {
156. **if**(fac[i] % 4 == 3 && (num[i] & 1))
157. **return** **true**;
158. }
159. **return** **false**;
160. }
162. T multi\_er(T a, T b, LL m)
163. {
164. T ans;
165. ans.p = (multi(a.p, b.p, m) + multi(multi(a.d, b.d, m), w, m)) % m;
166. ans.d = (multi(a.p, b.d, m) + multi(a.d, b.p, m)) % m;
167. **return** ans;
168. }
170. T power(T a, LL b, LL m)
171. {
172. T ans;
173. ans.p = 1;
174. ans.d = 0;
175. **while**(b)
176. {
177. **if**(b & 1)
178. {
179. ans = multi\_er(ans, a, m);
180. b--;
181. }
182. b >>= 1;
183. a = multi\_er(a, a, m);
184. }
185. **return** ans;
186. }
188. LL \_power(LL a, LL b)
189. {
190. LL ans = 1;
191. **while**(b)
192. {
193. **if**(b & 1)
194. {
195. ans = ans \* a;
196. b--;
197. }
198. b >>= 1;
199. a = a \* a;
200. }
201. **return** ans;
202. }
204. LL Legendre(LL a, LL p)
205. {
206. **return** quick\_mod(a, (p - 1) >> 1, p);
207. }
209. LL Work(LL n, LL p)
210. {
211. LL a = -1, t;
212. **while**(1)
213. {
214. a = rand() % p;
215. t = a \* a - n;
216. w = (t % p + p) % p;
217. **if**(Legendre(w, p) + 1 == p) **break**;
218. }
219. T tmp;
220. tmp.p = a;
221. tmp.d = 1;
222. T ans = power(tmp, (p + 1) >> 1, p);
223. **return** ans.p;
224. }
226. **void** Solve(LL n, LL p, LL &\_x, LL &\_y)
227. {
228. LL x = Work(n, p);
229. **if**(x >= p - x)
230. x = p - x;
231. LL t = p;
232. LL r = x;
233. LL limit = (LL)sqrt(1.0 \* p);
234. **while**(r > limit)
235. {
236. x = r;
237. r = t % x;
238. t = x;
239. }
240. \_x = r;
241. \_y = (LL)sqrt((p - r \* r));
242. **if**(\_x > \_y) swap(\_x, \_y);
243. }
245. **void** getData(Pair node[], **int** &\_cnt)
246. {
247. **for**(**int** i = 0; i < cnt; i++)
248. {
249. **if**(fac[i] == 2)
250. {
251. LL res = \_power(fac[i], num[i]>>1);
252. **if**(num[i] & 1)
253. node[\_cnt].x = res;
254. **else**
255. node[\_cnt].x = 0;
256. node[\_cnt].y = res;
257. \_cnt++;
258. **continue**;
259. }
260. **if**(fac[i] % 4 == 3)
261. {
262. LL res = \_power(fac[i], num[i]>>1);
263. node[\_cnt].x = 0;
264. node[\_cnt].y = res;
265. \_cnt++;
266. **continue**;
267. }
268. Solve(-1, fac[i], node[\_cnt].x, node[\_cnt].y);
269. \_cnt++;
270. **for**(**int** j = 1; j < num[i]; j++)
271. {
272. node[\_cnt].x = node[\_cnt - 1].x;
273. node[\_cnt].y = node[\_cnt - 1].y;
274. \_cnt++;
275. }
276. }
277. }
279. **void** dfs(**int** dept, **int** cnt, Pair ans)
280. {
281. **if**(dept == cnt - 1)
282. {
283. Set.insert(ans);
284. **return**;
285. }
286. **for**(**int** i = 0; i < 2; i++)
287. {
288. Pair \_ans;
289. \_ans.x = ans.x \* node[dept + 1].x + sign[i][0] \* ans.y \* node[dept + 1].y;
290. \_ans.y = ans.x \* node[dept + 1].y + sign[i][1] \* ans.y \* node[dept + 1].x;
291. **if**(\_ans.x < 0) \_ans.x = -\_ans.x;
292. **if**(\_ans.y < 0) \_ans.y = -\_ans.y;
293. **if**(\_ans.x > \_ans.y) swap(\_ans.x, \_ans.y);
294. **if**(Mem.find(\_ans) != Mem.end()) **return**;
295. Mem.insert(\_ans);
296. dfs(dept + 1, cnt, \_ans);
297. }
298. }
300. **void** Merge(Pair node[], **int** cnt)
301. {
302. Pair ans;
303. ans.x = node[0].x;
304. ans.y = node[0].y;
305. Mem.insert(ans);
306. dfs(0, cnt, ans);
307. }
309. **int** main()
310. {
311. LL n;
312. **while**(cin>>n)
313. {
314. **if**(n == 1)
315. {
316. puts("0 1");
317. **continue**;
318. }
319. Partion(n);
320. **if**(Filter())
321. {
322. puts("No solution");
323. **continue**;
324. }
325. Set.clear();
326. Mem.clear();
327. **int** \_cnt = 0;
328. getData(node, \_cnt);
329. Merge(node, \_cnt);
330. set<Pair>::iterator it;
331. **for**(it = Set.begin(); it != Set.end(); it++)
332. cout<<it->x<<" "<<it->y<<endl;
333. }
334. **return** 0;

}

[Bell数](http://blog.csdn.net/acdreamers/article/details/12309269)

**Bell数的定义：第n个Bell数表示集合{1,2,3,...,n}的划分方案数，即：B[0] = 1;**



**每一个Bell数都是第二类Stirling数的和，即：**



**第二类Stirling数的意义是：S(n,k)表示将n个物体划分成k个非空的不可辨别的（可以理解为盒子没有编号）集合的方法**

**数。很明显，每一个Bell是对应的第二类Stirling数之和。**

**ell数的指数生成函数是：**



**Bell三角形(构建方法类似于杨辉三角形)**

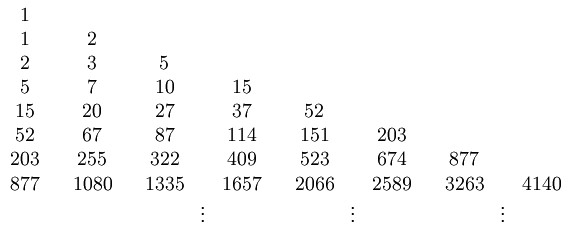
**Bell三角形的构造方法：**

**第一行第一个元素是1，即a[1][1] = 1**

**对于n>1，第n行第一项等于第n-1行最后一项，即a[n][1] = a[n-1][n-1];**

**对于m,n>1，第n行第m项等于它左边和左上方的两个数之和，即a[n][m] = a[n][m-1] + a[n-1][m-1];**

**如图：**



**可以看出，每行首项是贝尔数，每行之和是第二类Stirling数。**

**Bell还有两个重要的同余性质：**





**其中这里的p是不大于100的素数，这样，我们可以通过上面的性质来计算Bell数模小于100的素数值。**

**Bell数模素数p的周期为：**



**Bell数的预处理**

1. **void** Bell(**int** T[],**int** MOD)
2. {
3. B[0] = 1;
4. B[1] = 1;
5. T[0] = 1;
6. **for**(**int** i=2;i<N;i++)
7. {
8. T[i-1] = B[i-1];
9. **for**(**int** j=i-2;j>=0;j--)
10. T[j] = (T[j]+T[j+1])%MOD;
11. B[i] = T[0];
12. }
13. }

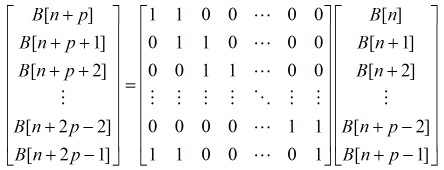
**题意：给定一个数n，范围是[1,2^31]，求Bell(n)(mod 95041567)**

**分析：注意95041567 = 31x37x41x43x47，那么我们先对每一个素数求出Bell(n)(mod p)，然后CRT合并即可。**

**在这里，我们有两种方法，第一种方法就是用**



**所以可以根据它构造50\*50的矩阵。如图：**



[第一类Stirling数和第二类Stirling](http://blog.csdn.net/acdreamers/article/details/8521134)

**第一类Stirling数 s(p,k)**

**s(p,k)的一个的组合学解释是：将p个物体排成k个非空循环排列的方法数。**

**s(p,k)的递推公式： s(p,k)=(p-1)\*s(p-1,k)+s(p-1,k-1) ,1<=k<=p-1**

**边界条件：s(p,0)=0 ,p>=1  s(p,p)=1  ,p>=0**

**递推关系的说明：**

**考虑第p个物品，p可以单独构成一个非空循环排列，这样前p-1种物品构成k-1个非空循环排列，方法数为s(p-1,k-1)；**

**也可以前p-1种物品构成k个非空循环排列，而第p个物品插入第i个物品的左边，这有(p-1)\*s(p-1,k)种方法。**

**第二类Stirling数 S(p,k)**

**S(p,k)的一个组合学解释是：将p个物体划分成k个非空的不可辨别的（可以理解为盒子没有编号）集合的方法数。**

**k!S(p,k)是把p个人分进k间有差别(如：被标有房号）的房间(无空房）的方法数。**

**S(p,k)的递推公式是：S(p,k)=k\*S(p-1,k)+S(p-1,k-1) ,1<= k<=p-1**

**边界条件：S(p,p)=1 ,p>=0    S(p,0)=0 ,p>=1**

**递推关系的说明：**

**考虑第p个物品，p可以单独构成一个非空集合，此时前p-1个物品构成k-1个非空的不可辨别的集合，方法数为S(p-1,k-1)；**

**也可以前p-1种物品构成k个非空的不可辨别的集合，第p个物品放入任意一个中，这样有k\*S(p-1,k)种方法。**

**第一类斯特林数和第二类斯特林数有相同的初始条件，但递推关系不同。**

**题意：给N个元素，让我们求K个环排列的方法数。**

**斯特林第一类数的第推公式：**

**S（N，0）=0; S（N，N）=1； S（0，0）=0； S（N，K）=S（N-1，K-1）+S（N-1，K）\*（N-1）；**

**这个公式的意思是：当前N-1个数构成K-1 个环的时候，加入第N个 ，N只能构成单环！—S（N-1，K-1）如果N-1个数构成K个环的时候，加入第N个，N可以任意加入，N-1内的一个环里，所以（N-1）\*S（N-1，K）这个题目里，因为不能破坏第1个门：所以 S（N，K）-S（N-1，K-1）才是能算构成K个环的方法数！就是去掉1自己成环的情况 。**

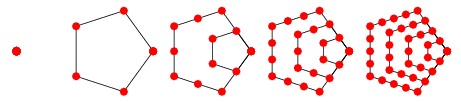
1. #define N 21
2. **\_\_int64** fac[N]={1,1};
3. **\_\_int64** stir[N][N];
5. **void** init()  {
6. **int** i, j;
7. **for**(i=2;i<N;i++)  fac[i]=i\*fac[i-1];
8. memset(stir,0,**sizeof**(stir));
9. stir[0][0]=0;
10. stir[1][1]=1;
11. **for**(i=2;i<N;i++)
12. **for**(j=1;j<=i;j++)
13. stir[i][j]=stir[i-1][j-1]+(i-1)\*stir[i-1][j];
14. }
15. **int** main()  {
16. init();**int** t;
17. scanf("%d",&t);
18. **while**(t--)  {
19. **int** n, k, i;
20. scanf("%d %d",&n,&k);
21. **\_\_int64** cnt=0;
22. **for**(i=1;i<=k;i++)
23. cnt+= stir[n][i] - stir[n-1][i-1];//注意：去掉1自己成环的
24. printf("%.4lf\n",1.0\*cnt/fac[n]);
25. }
26. **return** 0;

}

[五边形数定理](http://blog.csdn.net/acdreamers/article/details/12259815)

**设第n个五边形数为，那么，即序列为：1, 5, 12, 22, 35, 51, 70, ...**

**对应图形如下：**

****

**设五边形数的生成函数为，那么有：**

****

**以上是五边形数的情况。下面是关于五边形数定理的内容：**

**五边形数定理是一个由欧拉发现的数学定理，描述欧拉函数展开式的特性。欧拉函数的展开式如下：**





**欧拉函数展开后，有些次方项被消去，只留下次方项为1, 2, 5, 7, 12, ...的项次，留下来的次方恰为广义五边形数。**

**五边形数和分割函数的关系**

**欧拉函数的倒数是分割函数的母函数，亦即：**

**其中为k的分割函数。**

**上式配合五边形数定理，有：**



**在 n>0 时，等式右侧的系数均为0，比较等式二侧的系数，可得**



**因此可得到分割函数p(n)的递归式：**

**例如n=10时，有：所以，通过上面递归式，我们可以很快速地计算n的整数划分方案数p(n)了。**

1. **const** **int** N=100005;
2. **const** LL MOD=1000000007;
4. LL ans[N],tmp[N];
6. **void** Init()
7. {
8. **int** t=1000;
9. **for**(**int** i=-1000;i<=1000;i++)
10. tmp[i+t]=i\*(3\*i-1)/2;
11. ans[0]=1;
12. **for**(**int** i=1;i<N;i++)
13. {
14. ans[i]=0;
15. **for**(**int** j=1;j<=i;j++)
16. {
17. **if**(tmp[j+t]<=i)
18. {
19. **if**(j&1)  ans[i]+=ans[i-tmp[j+t]];
20. **else**     ans[i]-=ans[i-tmp[j+t]];
21. }
22. **else** **break**;
23. ans[i]=(ans[i]%MOD+MOD)%MOD;
24. **if**(tmp[t-j]<=i)
25. {
26. **if**(j&1) ans[i]+=ans[i-tmp[t-j]];
27. **else**    ans[i]-=ans[i-tmp[t-j]];
28. }
29. **else** **break**;
30. }
31. ans[i]=(ans[i]%MOD+MOD)%MOD;
32. }
33. }
34. **int** main()
35. {
36. **int** t,n;
37. Init();
38. cin>>t;
39. **while**(t--)
40. {
41. cin>>n;
42. cout<<ans[n]<<endl;
43. }
44. **return** 0;
45. }

**题意：问一个数n能被拆分成多少种方法，且每一种方法里数字重复个数不能超过k（等于k）。**

**分析递推式为**



1. **const** **int** N = 100005;
2. **const** **int** MOD = 1000000007;
4. **int** dp[N];
6. **void** Init()
7. {
8. dp[0] = 1;
9. **for**(**int** i=1;i<N;i++)
10. {
11. dp[i] = 0;
12. **for**(**int** j=1;;j++)
13. {
14. **int** t = (3\*j-1)\*j / 2;
15. **if**(t > i) **break**;
16. **int** tt = dp[i-t];
17. **if**(t+j <= i) tt = (tt + dp[i-t-j])%MOD;
18. **if**(j&1) dp[i] = (dp[i] + tt)%MOD;
19. **else**    dp[i] = (dp[i] - tt + MOD)%MOD;
20. }
21. }
22. }
24. **int** Work(**int** n,**int** k)
25. {
26. **int** ans = dp[n];
27. **for**(**int** i=1;;i++)
28. {
29. **int** t = k\*i\*(3\*i-1) / 2;
30. **if**(t > n) **break**;
31. **int** tt = dp[n-t];
32. **if**(t + i\*k <= n) tt = (tt + dp[n-t-i\*k])%MOD;
33. **if**(i&1) ans = (ans - tt + MOD)%MOD;
34. **else**    ans = (ans + tt)%MOD;
35. }
36. **return** ans;
37. }
39. **int** main()
40. {
41. Init();
42. **int** n,k,t;
43. scanf("%d",&t);
44. **while**(t--)
45. {
46. scanf("%d%d",&n,&k);
47. printf("%d\n",Work(n,k));
48. }
49. **return** 0;

}

[Catalan数](http://blog.csdn.net/acdreamers/article/details/7628667)

**Catalan数的定义：**

设表示用下面的方法把凸多边形区域分成三角形区域的方法数：在有n+1条边的凸多边形区域内通过插入在其中不相交的对角线而把它分成三角形区域。定义。则满足递推关系

这个递推关系的解是：，这里的叫做**Catalan数**。

那么上面的递推式的正确性我们可以简单描述一下即可：

**证明：**这里因为表示按照上述规则划分的三角形区域个数，那么我们随便选一条多边形的一条边作为基边，那么

     再在剩余的n-1个点中选一个点，我们把所选的一条边的两点分别与所选的那一点连接起来，那么多边形被划

     分成3部分，一部分有k+1条边，一部分有3条边，另一部分有n-k+1条边，那么这样就划分成了子问题了，所

     以按照这个思路可以证明递推式成立。

那么根据递推式是如何推出Catalan数的通项公式呢？

这里用到了生成函数：我们很容易写出的生成函数

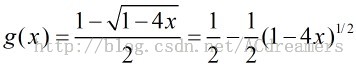
我们进一步计算



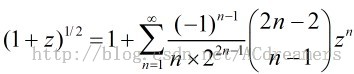
因为有：，所以进一步得到：

，由于

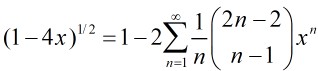
所以有：，解之得到：

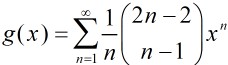
，另一个解不符合，舍去。

那么根据牛顿二项式有：



那么带入化简得到：



那么我们最终得到：

所以：，这就是**Catalan的推导过程**。

卡特兰数的应用

**1、括号化问题**

矩阵连乘：，依据乘法结合律，不改变其顺序，只用括号表示成对的乘积，问有几种括号化的方案？

**2、出栈次序问题**

一个栈(无穷大)的进栈序列为1,2,3,..n,有多少个不同的出栈序列?

**类似问题**

a、有2n个人排成一行进入剧场，入场费5元。其中只有n个人有一张5元钞票，另外n人只有10元钞票，剧院无其它钞票，问有多少中方法使得只要有10元的人买票，售票处就有5元的钞票找零？(将持5元者到达视作将5元入栈，持10元者到达视作使栈中某5元出栈)

b、n个1和n个0组成一个2n位的二进制数，要求从左到右扫描，0的累计数不小于1的累计数，求满足条件的的数。

c、12个人排成两排，每排必须是从矮到高排列，而且第二排比对应的第一排的人高，问排列方式有多少种？

我们先把这12个人从低到高排列,然后,选择6个人排在第一排,那么剩下的6个肯定是在第二排.用0表示对应的人在第一排,用1表示对应的人在第二排,那么含有6个0,6个1的序列,就对应一种方案.

比如000000111111就对应着

第一排：0 1 2 3 4 5 第二排：6 7 8 9 10 11 010101010101就对应着 第一排：0 2 4 6 8 10 第二排：1 3 5 7 9 11问题转换为，这样的满足条件的01序列有多少个。与情况b一样。

**3、给定节点组成二叉树的问题**

给定N个节点，能构成多少种形状不同的二叉树？

先取一个点作为顶点,然后左边依次可以取0至N-1个相对应的,右边是N-1到0个,两两配对相乘,就是h(0)\*h(n-1) + h(2)\*h(n-2) +  + h(n-1)h(0)=h(n))能构成h（N）个

**4.n\*n棋盘从左下角走到右上角而不穿过主对角线的走法**

**5.n个+1和n个-1构成的2n项序列，其部分和总满足：的序列的个数。**

Catalan数的高精度处理：利用递归式： h(n)=((4\*n-2)/(n+1))\*h(n-1)

1. **int** a[105][105];    //大数卡特兰数
2. **int** b[105];         //卡特兰数的长度
4. **void** catalan()  //求卡特兰数
5. {
6. **int** i,j,len,carry,temp;
7. a[1][0]=b[1]=1;
8. len=1;
9. **for**(i=2;i<=100;i++)
10. {
11. **for**(j=0;j<len;j++)    //乘法
12. a[i][j]=a[i-1][j]\*(4\*(i-1)+2);
13. carry=0;
14. **for**(j=0;j<len;j++)    //处理相乘结果
15. {
16. temp=a[i][j]+carry;
17. a[i][j]=temp%10;
18. carry=temp/10;
19. }
20. **while**(carry)    //进位处理
21. {
22. a[i][len++]=carry%10;
23. carry/=10;
24. }
25. carry=0;
26. **for**(j=len-1;j>=0;j--) //除法
27. {
28. temp=carry\*10+a[i][j];
29. a[i][j]=temp/(i+1);
30. carry=temp%(i+1);
31. }
32. **while**(!a[i][len-1])     //高位零处理
33. len--;
34. b[i]=len;
35. }
36. }
38. **int** main()
39. {
40. **int** i,n;
41. catalan();
42. **while**(~scanf("%d",&n),n)
43. {
44. **for**(i=b[n]-1;i>=0;i--)
45. printf("%d",a[n][i]);
46. printf("\n");
47. }
48. **return** 0;

}

数学中的若干定理和数列前n项

1. 求N！在K进制下的位数，即计算：

2.计算弧长公式：

3.部分重要积分公式：

4.解九连环公式：

5.证明：若，那么

6.给定正整数n和x，求

分析：牛顿迭代法，利用，这里从开始迭代，迭代次数越多，结果就越精

卡特兰数

1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012,

742900, 2674440, 9694845, 35357670, 129644790, 477638700, 1767263190,

6564120420, 24466267020, 91482563640, 343059613650, 1289904147324, ...

伯努利数

B0 = 1, B1 = -1/2

B2 = 1/6, B3 = 0

B4 = -1/30, B5 = 0

B6 = 1/42, B7 = 0,

B8 = -1/30, B9 = 0,

B10 = 5/66, B11 = 0,

B12 = -691/2730,B13 = 0,

B14= 7/6, B15 = 0,

B16 = -3617/510,B17 = 0,

B18 = 43867/798,B19 = 0,

B20 = -174611/330, B21 = 0 ……

Bell数

1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975,

第一类Stirling数

1, -1, 1, 2, -3, 1, -6, 11, -6, 1, 24, -50,

35, -10, 1, -120, 274, -225, 85, -15, 1, 720,

-1764, 1624, -735, 175, -21, 1

第二类Stirling数

1, 1, 1, 1, 3, 1, 1, 7, 6, 1, 1, 15, 25, 10,

1, 1, 31, 90, 65, 15, 1, 1, 63, 301, 350, 140, 21, 1, 1,

127, 966, 1701, 1050, 266, 28, 1, 1, 255, 3025, 7770,

6951, 2646, 462, 36, 1, 1, 511, 9330, 34105, 42525, 22827,

[Fibonacci数列的幂和](http://blog.csdn.net/acdreamers/article/details/23039571)

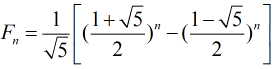
**题意：**给定和，其中，，求  的值。

**分析：**嗯，这道题貌似有难度，如果比较小的话我们可以构造矩阵，实际上这样做也挺麻烦的。

     以前我们做一个大Fibonacci数列模一个大素数都是用矩阵，当然这里素数满足条件：**5是模这个素数的二**

**次剩余，**那么现在要求不要用矩阵来计算这个结果呢？那就是今天我要讨论的问题，本题也是基于这种思路。

     本题我们可以直接利用Fibonacci数列的公式进行计算。因为我们知道Fibonacci数列的公式为：



     虽然公式中含有根号，但是我们知道是一个整数，而且5是模1000000009的二次剩余。那么可以通过逆元

     和二次剩余的转化来做。比如就可以用中的来代替。

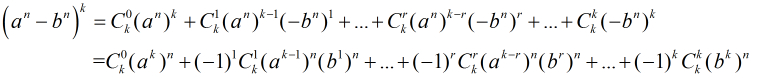
     为了方便表示，我们令：



     那么得到



     把按照二项式展开得到



     对于每一个都这样表示，那么相同的合并后是等比数列，比如对于所有系数为合并后为

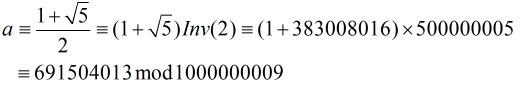
， 其中

     所以到了这里本题就明确了，枚举每一个从**0**到，依次计算即可。

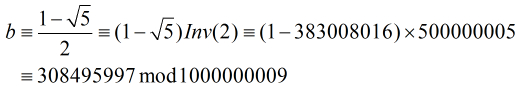
     当然对于，我们可以阶乘预处理然后求逆元，至于和同样预处理，这样时间少很多。

     可以看出本题条件好在1000000009是素数，而且5是模1000000009的二次剩余

     经计算2模1000000009的逆元是**500000005**，而的一个解为**383008016**。也就是说对于有



     同理，对于有



     嗯，貌似还有一个问题没有处理，t = 1时咋办？ 这个很简单啦，不用等比求和公式即可。其实吧，这里面

     我们还可以注意到，也可以进一步优化，读者自己体会，就说到这里。

**代码：**

1. #include <iostream>
2. #include <string.h>
3. #include <stdio.h>
5. **using** **namespace** std;
6. **typedef** **long** **long** LL;
7. **const** **int** N = 100005;
8. **const** LL MOD = 1000000009;
10. LL fac[N],A[N],B[N];
12. **void** Init()
13. {
14. fac[0] = 1;
15. **for**(**int** i=1; i<N; i++)
16. fac[i] = fac[i-1] \* i % MOD;
17. A[0] = B[0] = 1;
18. **for**(**int** i=1; i<N; i++)
19. {
20. A[i] = A[i-1] \* 691504013 % MOD;
21. B[i] = B[i-1] \* 308495997 % MOD;
22. }
23. }
25. LL quick\_mod(LL a,LL b,LL MOD)
26. {
27. LL ans = 1;
28. a %= MOD;
29. **while**(b)
30. {
31. **if**(b & 1)
32. {
33. ans = ans \* a % MOD;
34. b--;
35. }
36. b >>= 1;
37. a = a \* a % MOD;
38. }
39. **return** ans;
40. }
42. LL Solve(LL n,LL k)
43. {
44. LL ans = 0;
45. **for**(**int** r=0; r<=k; r++)
46. {
47. LL t = A[k-r] \* B[r] % MOD;
48. LL x = fac[k];
49. LL y = fac[k-r] \* fac[r] % MOD;
50. LL c = x \* quick\_mod(y,MOD-2,MOD) % MOD;
51. LL tmp = t \* (quick\_mod(t,n,MOD) - 1) % MOD \* quick\_mod(t-1,MOD-2,MOD) % MOD;
52. **if**(t == 1) tmp = n % MOD;
53. tmp = tmp \* c % MOD;
54. **if**(r & 1) ans -= tmp;
55. **else**      ans += tmp;
56. ans %= MOD;
57. }
58. LL m = quick\_mod(383008016,MOD-2,MOD);
59. ans = ans \* quick\_mod(m,k,MOD) % MOD;
60. ans = (ans % MOD + MOD) % MOD;
61. **return** ans;
62. }
64. **int** main()
65. {
66. **int** T;
67. LL n,k;
68. Init();
69. scanf("%d",&T);
70. **while**(T--)
71. {
72. cin>>n>>k;
73. cout<<Solve(n,k)<<endl;
74. }
75. **return** 0;

}

[Fib数模n的循环节](http://blog.csdn.net/acdreamers/article/details/10983813)

**我们知道Fibonacci数列，现在我们来求一个Fib数模n的循环节的长度。**

对于一个正整数n，我们求Fib数模n的循环节的长度的方法如下：

  （1）把n素因子分解，即

    （2）分别计算Fib数模每个的循环节长度，假设长度分别是

    （3）那么Fib模n的循环节长度

从上面三个步骤看来，貌似最困难的是第二步，那么我们如何求Fib模的循环节长度呢？

     这里有一个优美的定理：Fib数模的最小循环节长度等于，其中表示Fib数模素数的最小循环节长度。可以看出我们现在最重要的就是求

对于求我们利用如下定理：

   如果5是模的二次剩余，那么循环节的的长度是的因子，否则，循环节的长度是的因子

顺便说一句，对于小于等于5的素数，我们直接特殊判断，loop(2)=3,loop(3)=8,loop(5)=20。

那么我们可以先求出所有的因子，然后用矩阵快速幂来一个一个判断，这样时间复杂度不会很大。

**模板代码：**

1. #include <iostream>
2. #include <string.h>
3. #include <algorithm>
4. #include <stdio.h>
5. #include <math.h>
7. **using** **namespace** std;
8. **typedef** unsigned **long** **long** LL;
10. **const** **int** M = 2;
12. **struct** Matrix
13. {
14. LL m[M][M];
15. };
17. Matrix A;
18. Matrix I = {1,0,0,1};
20. Matrix multi(Matrix a,Matrix b,LL MOD)
21. {
22. Matrix c;
23. **for**(**int** i=0; i<M; i++)
24. {
25. **for**(**int** j=0; j<M; j++)
26. {
27. c.m[i][j] = 0;
28. **for**(**int** k=0; k<M; k++)
29. c.m[i][j] = (c.m[i][j]%MOD + (a.m[i][k]%MOD)\*(b.m[k][j]%MOD)%MOD)%MOD;
30. c.m[i][j] %= MOD;
31. }
32. }
33. **return** c;
34. }
36. Matrix power(Matrix a,LL k,LL MOD)
37. {
38. Matrix ans = I,p = a;
39. **while**(k)
40. {
41. **if**(k & 1)
42. {
43. ans = multi(ans,p,MOD);
44. k--;
45. }
46. k >>= 1;
47. p = multi(p,p,MOD);
48. }
49. **return** ans;
50. }
52. LL gcd(LL a,LL b)
53. {
54. **return** b? gcd(b,a%b):a;
55. }
57. **const** **int** N = 400005;
58. **const** **int** NN = 5005;
60. LL num[NN],pri[NN];
61. LL fac[NN];
62. **int** cnt,c;
64. **bool** prime[N];
65. **int** p[N];
66. **int** k;
68. **void** isprime()
69. {
70. k = 0;
71. memset(prime,**true**,**sizeof**(prime));
72. **for**(**int** i=2; i<N; i++)
73. {
74. **if**(prime[i])
75. {
76. p[k++] = i;
77. **for**(**int** j=i+i; j<N; j+=i)
78. prime[j] = **false**;
79. }
80. }
81. }
83. LL quick\_mod(LL a,LL b,LL m)
84. {
85. LL ans = 1;
86. a %= m;
87. **while**(b)
88. {
89. **if**(b & 1)
90. {
91. ans = ans \* a % m;
92. b--;
93. }
94. b >>= 1;
95. a = a \* a % m;
96. }
97. **return** ans;
98. }
100. LL legendre(LL a,LL p)
101. {
102. **if**(quick\_mod(a,(p-1)>>1,p)==1) **return** 1;
103. **else**                           **return** -1;
104. }
106. **void** Solve(LL n,LL pri[],LL num[])
107. {
108. cnt = 0;
109. LL t = (LL)sqrt(1.0\*n);
110. **for**(**int** i=0; p[i]<=t; i++)
111. {
112. **if**(n%p[i]==0)
113. {
114. **int** a = 0;
115. pri[cnt] = p[i];
116. **while**(n%p[i]==0)
117. {
118. a++;
119. n /= p[i];
120. }
121. num[cnt] = a;
122. cnt++;
123. }
124. }
125. **if**(n > 1)
126. {
127. pri[cnt] = n;
128. num[cnt] = 1;
129. cnt++;
130. }
131. }
133. **void** Work(LL n)
134. {
135. c = 0;
136. LL t = (LL)sqrt(1.0\*n);
137. **for**(**int** i=1; i<=t; i++)
138. {
139. **if**(n % i == 0)
140. {
141. **if**(i \* i == n) fac[c++] = i;
142. **else**
143. {
144. fac[c++] = i;
145. fac[c++] = n / i;
146. }
147. }
148. }
149. }
151. LL find\_loop(LL n)
152. {
153. Solve(n,pri,num);
154. LL ans=1;
155. **for**(**int** i=0; i<cnt; i++)
156. {
157. LL record=1;
158. **if**(pri[i]==2)
159. record=3;
160. **else** **if**(pri[i]==3)
161. record=8;
162. **else** **if**(pri[i]==5)
163. record=20;
164. **else**
165. {
166. **if**(legendre(5,pri[i])==1)
167. Work(pri[i]-1);
168. **else**
169. Work(2\*(pri[i]+1));
170. sort(fac,fac+c);
171. **for**(**int** k=0; k<c; k++)
172. {
173. Matrix a = power(A,fac[k]-1,pri[i]);
174. LL x = (a.m[0][0]%pri[i]+a.m[0][1]%pri[i])%pri[i];
175. LL y = (a.m[1][0]%pri[i]+a.m[1][1]%pri[i])%pri[i];
176. **if**(x==1 && y==0)
177. {
178. record = fac[k];
179. **break**;
180. }
181. }
182. }
183. **for**(**int** k=1; k<num[i]; k++)
184. record \*= pri[i];
185. ans = ans/gcd(ans,record)\*record;
186. }
187. **return** ans;
188. }
190. **void** Init()
191. {
192. A.m[0][0] = 1;
193. A.m[0][1] = 1;
194. A.m[1][0] = 1;
195. A.m[1][1] = 0;
196. }
198. **int** main()
199. {
200. LL n;
201. Init();
202. isprime();
203. **while**(cin>>n)
204. cout<<find\_loop(n)<<endl;
205. **return** 0;

}