

原创

Product Oriented Recurrence （矩阵快速幂+欧拉扩展公式）

2019-06-13 21:27:03    -Y\_-Y\_-    阅读数 79    更多

编辑

版权声明：本文为博主原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接和本声明。  
本文链接：[https://blog.csdn.net/weixin\\_44410512/article/details/91897500](https://blog.csdn.net/weixin_44410512/article/details/91897500)

题目链接

<https://cn.vjudge.net/problem/CodeForces-1182E>  
题目大意

$$f_x = c^{2x-6} \cdot f_{x-1} \cdot f_{x-2} \cdot f_{x-3} \text{ for } x \geq 4.$$

输入  $n, f_1, f_2, f_3, c$  ( $4 \leq n \leq 1018, 1 \leq f_1, f_2, f_3, c \leq 109$ ).  
输出  $f_n \% (10^9 + 7)$ .

思路

```
1  我们可以将fn拆乘 （x个f1相乘）乘以 （y个f2相乘）乘以 （z个f3相乘）乘以 pow（c,k）
2
3      f3  f2  f1
4  1   0   0   1
5  2   0   1   0
6  3   1   0   0
7  4   1   1   1
8  5   2   2   1
9  6   4   3   2
10 7   7   6   4
11
12 从第四行开始
13 第n行的f1系数等于 第(n-1)行f1系数+第(n-2)行f1系数+第(n-3)行f1系数
14 第n行的f2系数等于 第(n-1)行f2系数+第(n-2)行f2系数+第(n-3)行f2系数
15 第n行的f3系数等于 第(n-1)行f3系数+第(n-2)行f3系数+第(n-3)行f3系数
16
17 以f1为例
18 矩阵可以写成
19 1 1 1      4 0 0
20 1 0 0 * 2 0 0
21 0 1 1      1 0 0
22
23 接下来就是求 pow(c,k);
24
25 1   1
26 2   1
27 3   1
28 4   pow(c,2)
29 5   pow(c,4)*pow(c,2)
30 6   pow(c,6)*pow(c,4)*pow(c,2)*pow(c,2)
31 7   pow(c,8)*pow(c,4)*pow(c,2)*pow(c,2) * pow(c,4)*pow(c,2) * pow(c,2)
32
33 从第四行开始
34 第n行为 第(n-1)行结果*第(n-2)行结果*第(n-3)行结果*pow(c,2*n-6)
35
36 矩阵可以写成
37 1 1 1 2 -6      14 0 0 0 0
38 1 0 0 0 0      6 0 0 0 0
39 0 1 0 0 0 * 2 0 0 0 0
40 0 0 0 1 1      7 0 0 0 0
41 0 0 0 0 1      1 0 0 0 0
42
43
```

AC code

编程语言大PK，你选谁？

关闭

```

1  #include <algorithm>
2  #include <cstdio>
3  #include <cstring>
4  #include <iostream>
5  using namespace std;
6  #define ll long long
7  const ll MOD = 1e9 + 7;
8  struct matrix{// 矩阵快速幂
9      ll m[5][5];
10 };
11 ll pow(ll x,ll n,ll mod){// 快速幂
12     ll res=1;
13     while(n>0){
14         if(n%2==1){
15             res=res*x;
16             res=res%mod;
17         }
18         x=x*x;
19         x=x%mod;
20         n>>=1;
21     }
22     return res;
23 }
24
25 matrix matrix_multi(matrix a,matrix b){// 矩阵相乘
26     matrix tmp;
27     for(int i=0;i<5;i++){
28         for(int j=0;j<5;j++){
29             tmp.m[i][j]=0;
30             for(int k=0;k<5;k++){
31                 tmp.m[i][j]=(tmp.m[i][j])% (MOD-1) + (a.m[i][k]*b.m[k][j]+MOD-1)% (MOD-1)) % (MOD-1);
32             }
33         }
34     }
35     return tmp;
36 }
37
38 matrix matrix_pow(matrix a,matrix b,ll n){// 矩阵快快速幂
39     while(n>0){
40         if(n&1) b=matrix_multi(a,b);
41         a=matrix_multi(a,a);
42         n>>=1;
43     }
44     return b;
45 }
46
47 ll ff3(ll f3, ll n){
48     if(n==4) return pow(f3,1,MOD);
49     if(n==5) return pow(f3,2,MOD);
50     if(n==6) return pow(f3,4,MOD);
51     matrix a,b;
52     memset(a.m,0,sizeof a.m);
53     memset(b.m,0,sizeof b.m);
54     a.m[0][0]=1; a.m[0][1]=1; a.m[0][2]=1;
55     a.m[1][0]=1; a.m[1][1]=0; a.m[1][2]=0;
56     a.m[2][0]=0; a.m[2][1]=1; a.m[2][2]=0;
57
58     b.m[0][0]=4; b.m[0][1]=0; b.m[0][2]=0;
59     b.m[1][0]=2; b.m[1][1]=0; b.m[1][2]=0;
60     b.m[2][0]=1; b.m[2][1]=0; b.m[2][2]=0;
61
62     matrix ans=matrix_pow(a,b,n-6);
63     return pow(f3,ans.m[0][0],MOD);
64 }
65
66 ll ff2(ll f2,ll n){
67     if(n==4) return pow(f2,1,MOD);
68     if(n==5) return pow(f2,2,MOD);
69     if(n==6) return pow(f2,3,MOD);
70     matrix a,b;
71     memset(a.m,0,sizeof a.m);
72     memset(b.m,0,sizeof b.m);

```

```
71     a.m[0][0]=1; a.m[0][1]=1; a.m[0][2]=1;
72     a.m[1][0]=1; a.m[1][1]=0; a.m[1][2]=0;
73     a.m[2][0]=0; a.m[2][1]=1; a.m[2][2]=0;
74
75     b.m[0][0]=3; b.m[0][1]=0; b.m[0][2]=0;
76     b.m[1][0]=2; b.m[1][1]=0; b.m[1][2]=0;
77     b.m[2][0]=1; b.m[2][1]=0; b.m[2][2]=0;
78
79     matrix ans=matrix_pow(a,b,n-6);
80     return pow(f2,ans.m[0][0],MOD);
81 }
82
83 ll ff1(ll f1,ll n){
84     if(n==4) return pow(f1,1,MOD);
85     if(n==5) return pow(f1,1,MOD);
86     if(n==6) return pow(f1,2,MOD);
87     matrix a,b;
88     memset(a.m,0,sizeof a.m);
89     memset(b.m,0,sizeof b.m);
90     a.m[0][0]=1; a.m[0][1]=1; a.m[0][2]=1;
91     a.m[1][0]=1; a.m[1][1]=0; a.m[1][2]=0;
92     a.m[2][0]=0; a.m[2][1]=1; a.m[2][2]=0;
93
94     b.m[0][0]=2; b.m[0][1]=0; b.m[0][2]=0;
95     b.m[1][0]=1; b.m[1][1]=0; b.m[1][2]=0;
96     b.m[2][0]=1; b.m[2][1]=0; b.m[2][2]=0;
97
98     matrix ans=matrix_pow(a,b,n-6);
99     return pow(f1,ans.m[0][0],MOD);
100 }
101
102 ll cc(ll c,ll n){
103
104     if(n==4) return pow(c,2,MOD);
105     if(n==5) return pow(c,6,MOD);
106     if(n==6) return pow(c,14,MOD);
107     matrix a,b;
108     memset(a.m,0,sizeof a.m);
109     memset(b.m,0,sizeof b.m);
110     b.m[0][0]=14;
111     b.m[1][0]=6;
112     b.m[2][0]=2;
113     b.m[3][0]=7;
114     b.m[4][0]=1;
115
116     a.m[0][0]=1; a.m[0][1]=1; a.m[0][2]=1; a.m[0][3]=2; a.m[0][4]=-6;
117     a.m[1][0]=1; a.m[1][1]=0; a.m[1][2]=0; a.m[1][3]=0; a.m[1][4]=0;
118     a.m[2][0]=0; a.m[2][1]=1; a.m[2][2]=0; a.m[2][3]=0; a.m[2][4]=0;
119     a.m[3][0]=0; a.m[3][1]=0; a.m[3][2]=0; a.m[3][3]=1; a.m[3][4]=1;
120     a.m[4][0]=0; a.m[4][1]=0; a.m[4][2]=0; a.m[4][3]=0; a.m[4][4]=1;
121
122     matrix ans=matrix_pow(a,b,n-6);
123     return pow(c,ans.m[0][0],MOD);
124 }
125
126 void solve(){
127     ll n,f1,f2,f3,c;
128     scanf("%lld %lld %lld %lld %lld", &n, &f1, &f2, &f3, &c);
129     if(n==1){
130         printf("%lld\n", f1);
131         return ;
132     }
133     if(n==2){
134         printf("%lld\n", f2);
135         return ;
136     }
137     if(n==3){
138         printf("%lld\n", f3);
139         return ;
140     }
141     ll ans=1;
```

```
142     anss=( anss * ff1(f1,n))%MOD;
143     anss=( anss * ff2(f2,n))%MOD;
144     anss=( anss * ff3(f3,n))%MOD;
145     anss=( anss * cc(c,n))%MOD;
146     printf("%lld\n", anss);
147     return ;
148 }
149
150 int main(){
151     solve();
152     return 0;
153 }
154
```

为什么要在矩阵相乘里面要MOD-1呢?

因为矩阵算出来的  $x, y, z, k$  意思是  $f_1^x, f_2^y, f_3^z, c^k$

又因欧拉定理的推论:

若正整数  $a, n$  互质, 那么对于任意正整数  $b$ , 有  $a^b \equiv a^{b \bmod \varphi(n)} \pmod{n}$

所以最终结果是

$$(f_1^{x \% MOD-1} \times f_2^{y \% MOD-1} \times f_3^{z \% MOD-1} \times c^{k \% MOD-1}) \% MOD$$

有 0 个人打赏

文章最后发布于: 2019-06-13 21:27:03

©2019 CSDN 皮肤主题: 大白 设计师: CSDN官方博客

编程语言大PK，你选谁？

关闭