快速幂

信智学院　陈琰宏

基本概念

整数集合： $Z=\{\ldots,-2,-1,0,1,2,\ldots\}$

自然数集合： $N=\{0,1,2,\ldots\}$

整除： 设a是非零整数，b是整数。如果存在一个整数q，使得 b=a*q，那么就说b可被a整除，记做$a|b$，且称b是a的倍数，a是b的约数（因子）。

基本性质：

1. 如果a|b，且b|c，那么a|c。
2. a|b且a|c，等价于任意的整数x和y，有a|(b*x+c*y)。
3. 设$m \neq 0$，那么$a|b$ 等价于$(m*a)|(m*b)$。
4. 设整数x和y满足：a*x+b*y=1，且a|n，b|n，那么（a*b）|n。
5. 若$b=q*d+c$，那么 $d|b$的充要条件时$d|c$。

同余

一、概念

一般地，设n为正整数，a、b为整数，如果a和b被n除后余数相同，那么称a和b模n同余，记作a≡b(mod n）。如果a和b被n除后余数不同，那么称a和b模n不同余。

二、同余与整除的关系

$$a \equiv b(mod\, n) \leftrightarrow n|a-b$$

a-b 也能被n整除

同余

二、同余与整除的关系

$$a \equiv b(mod\ n) \leftrightarrow n|a-b$$

设a、b被n除后商分别为q、q',余数分别为r、r'，则

a = nq + r　　b = nq' + r'

其中0 <= r < n, 0 <= r' < n.

若 a ≡ b (mod n),

则 r = r', (a − b) = n(q − q'),

所以n | a − b.

反过来，n | a - b = n(q - q') + r - r',则n | r - r',因为

-n < r - r' < n,所以只能是r = r‘，所以a和b模n同余.

同余

三、性质

从同余的概念和一些简单的探究，同余式$a \equiv b(\mathrm{mod}\ n)$与等式$a = b$又许多类似的性质

1、若$a \equiv b(\mathrm{mod}\ n), c \equiv d(\mathrm{mod}\ n)$,则

(1) $a + c \equiv b + d(\mathrm{mod}\ n)$

(2) $ac \equiv bd(\mathrm{mod}\ n)$

(3) $ka \equiv kb(\mathrm{mod}\ n)$, k为任意整数

(4) $a^m \equiv b^m(\mathrm{mod}\ n)$， m为正整数

2、若$ab \equiv ac(\mathrm{mod}\ n), (a, n) = 1$,则$b \equiv c(\mathrm{mod}\ n)$

1 [1314] 指数取模

输入整数m, n, k, 求m^n mod k的值。m, n, k*k为长整型范围内的自然数。

输入一行3个整数，分别为m, n,

输出一行一个整数，表示结果。

分析

如何求 $3^{89} \bmod 7$?

$3^1 \equiv 3(mod\ 7)$

$3^2 \equiv 3^2(mod\ 7) \equiv 2(mod\ 7)$

$3^4 \equiv (3^2)^2(mod\ 7) \equiv 2^2(mod\ 7) \equiv 4$

$3^8 \equiv (3^4)^2(mod\ 7) \equiv 4^2(mod\ 7) \equiv 2$

$3^{16} \equiv (3^8)^2(mod\ 7) \equiv 2^2(mod\ 7) \equiv 4$

$3^{32} \equiv (3^{16})^2(mod\ 7) \equiv 4^2(mod\ 7) \equiv 2$

… …

$3^{89} \equiv (3^{64}) * (3^{32}) * (3^{16}) * (3^8) * (3^1)(mod\ 7)$

$\equiv 4 * 4 * 2 * 3(mod\ 7) \equiv 96(mod\ 7)$

$\equiv 5(mod\ 7)$

所以，首先将n分解成2的幂次方，存放在一个数组r中，r[i]=1表示有 m^i这一项。然后用递推的方法从小到大逐个求出 $m^i \bmod k$ 的值（j从i到1）存放在数组d中。

$$x\hat{}y = \begin{cases} 1 & \text{如果} y = 0 \\ x * x\hat{}(y-1) & \text{如果} y\text{是奇数} \\ (x\hat{}2)\hat{}(y\ div\ 2) & \text{如果} y\text{是偶数} \end{cases}$$

$$3^{89} = (3^{44})^2 * 3 = \ldots = ((((((3^2) * 3)^2 * 3)^2)^2)^2 * 3$$

```cpp
#include<bits/stdc++.h>
using namespace std;
typedef long long LL;
LL m,n,k;
int main()
{
    cin>>m>>n>>k;
    LL ans=1;
    for(;n;n>>=1,m=m*m%k)
        if(n&1)
            ans=ans*m%k;
    cout<<ans;
    return 0;
}
```

```cpp
LL kasumi(LL x,LL mi){
    LL res=1;
    while(mi){
        if(mi&1){
            res=(res*x)%m;

        }
        x=(x*x)%m;
        mi>>=1;
    }
    return res;
}
int main(){
    cin>>n>>c>>m;
    cout<<kasumi(n,c);
    return 0;
}
```

```cpp
#include<bits/stdc++.h>
using namespace std;
typedef long long LL;
LL m, c,n;
```

2 [3647 1591] 等差等比、序列的第k个数

陈晖在学等差数列和等比数列，当已知前三项时，就可以知道是等差数列还是等比数列。现在给你序列的前三项，这个序列要么是等差序列，要么是等比序列，你能求出第 k 项的值吗。 如果第 k 项的值太大，对 200907 取模。

输入第一行一个整数 T，表示有 T 组测试数据；

对于每组测试数据，输入前三项 a,b,c，然后输入 k。

对于全部数据，$1<=T<=100, 1<=a<=b<=c<=10^9, 1<=k<=10^9$

[3647] 陈晖的等差等比

输出格式
对于每组数据输出第 k 项的值，对 200907 取模。
输入样例
2
1 2 3 5
1 2 4 5
输出样例
5
16

分析

如何判断是等差还是等比数列；

对于等差数列，$a_i+a_{i+2}=2*a_{i+1}$，就是等差数列，根据通项$a_n=a_1+(n-1)*d$，就可以求出第n项。

对于等比数列，$a_i/a_{i-1}=a_{i+1}/a_i$，求第n项时，$a_n=a_1*q^{n-1}$，求q的n-1次幂，需要使用快速幂

[3647] 陈眸的等差等比

```
int main(){
    int t;
    LL a,b,c,k,ans;
    cin>>t;
    while(t--){
        cin>>a>>b>>c>>k;
        if(b-a==c-b){//等差
            ans=(a+(b-a)*(k-1))%mod;
        }
        else{//等比
            ans=a*kasumi(b/a,k-1)%mod;
        }
        cout<<ans<<"\n";
    }
}
```

[3647] 陈眸的等差等比

```cpp
#include<bits/stdc++.h>
using namespace std;
typedef long long LL;
const LL mod=200907;
LL kasumi(LL x,LL mi){
    LL res=1;
    while(mi){
        if(mi&1){
            res=(res*x)%mod;

        }
        x=(x*x)%mod;
        mi>>=1;
    }
    return res;
}
```

3 [6382] 快速幂求逆元

给定 n 组 ai,pi，其中 pi 是质数，求 ai 模 pi 的乘法逆元，若逆元不存在则输出 impossible。

注意：请返回在 0~ p-1之间的逆元。

乘法逆元的定义

若整数 b，m 互质，并且对于任意的整数 a，如果满足 b|a，则存在一个整数 x，使得 $\frac{a}{b} \equiv a \times x (mod\, m)$ ，则称 x 为 b 的模 m 乘法逆元，记为$b^{-1}(mod\, m)$ 。

b 存在乘法逆元的充要条件是 b 与模数 m 互质。当模数 m 为质数时，b^{m-2} 即为 b 的乘法逆元。

分析

定义:即已知 b 与 m 互质 且 $b|a$ 则求一个x 使得

$a/b \equiv a * x \mod m$

[注] 简单定义 即 $b * x \equiv 1 \mod m$ 且 b 与 m 互质 则 x 为 b 的逆元

由费马小定理可知

$$b^{m-1} \equiv 1 \mod m$$

费马小定理：假如 a 是一个整数， p 是一个质数，那么 $a^p - a$ 是 p 的倍数，也可以表示为 $a^p \equiv a \ (mod\ p)$。如果 a 不是 p 的倍数的话，这个定理可以改写成我们常看的形式：$a^{p-1} \equiv 1 \ (mod\ p)$ 。

费马小定理证明

$$a^p \equiv a \pmod p$$
$$a^{p-1} \equiv 1 \pmod p$$

证明1. $\{1, 2, \cdots, p-1\}$

$\{a, 2a, \cdots, (p-1)a\}$ 互不相同.

若 $a \cdot i = a \cdot j \pmod p$, $1 \le i, j \le p-1$

$\Rightarrow a(i-j) \equiv 0 \pmod p$

$\Rightarrow p \mid a(i-j)$. 但 $p \nmid i-j$, $p \nmid a$ 矛盾!

$\Rightarrow (p-1)! \equiv a^{p-1}(p-1)! \pmod p$.

费马小定理证明

$$a^p \equiv a \pmod{p}$$
$$a^{p-1} \equiv 1 \pmod{p}$$

$\Rightarrow \quad (p-1)! \equiv a^{p-1}(p-1)! \pmod{p}$.

注意到 $(p, (p-1)!) = 1$.

$\Rightarrow \quad a^{p-1} \equiv 1 \pmod{p}$.

由费马小定理可知

$$b^{m-1} \equiv 1 \mod m$$

而

$$a/b \equiv a * x \mod m$$

联立以上两式，得:

$$a/b * b^{m-1} \equiv a * x \mod m$$

即为

$$a * b^{m-2} \equiv a * x \mod m$$

$$a * b^{m-2} \equiv a * x \mod m$$

而$b|a$,且b与m互质,因此对于b来说,一定存在一个a也与m互质,故而a可以在两边同时约去

因此

$$x \equiv b^{m-2} \mod m$$

b=3,m=5, x=3^(5-2) %5=2

代码

```
5  LL qmi(int a, int b, int p)
6  {
7      LL res = 1;
8      while(b){
9          if(b & 1) res = res * a % p;
10         a = (LL)a * a % p;
11         b >>= 1;
12     }
13     return res;
14 }
```

```
16 int main()
17 {
18     int n; cin >> n;
19     while(n --){
20         int a, p;
21         cin >> a >> p;
22         if(a % p == 0)
23             puts("impossible");
24         else
25             cout << qmi(a, p - 2, p) <<"\n";
26     }
27     return 0;
28 }
```

4 [3649] 越狱

监狱有连续编号为 1 到 n 的 n 个房间，每个房间关押一个犯人。有 m 种宗教，每个犯人可能信仰其中一种。如果相邻房间的犯人信仰的宗教相同，就可能发生越狱。求有多少种状态可能发生越狱。

输入两个整数 m 和 n。对于全部数据，$1 \leqslant m \leqslant 10^8, 1 \leqslant n \leqslant 10^{12}$。

输出可能越狱的状态数，对 100003 取余。

输入	输出
2 3	6

分析

考虑 n 个犯人，m种宗教，如何安排不会导致犯罪。

第一个位置可以有 m 个选择，则与第一个相邻的第二个位置就只有 $m-1$ 中选择。

考虑第 i 个位置，则为了不和他左侧的 $i-1$ 位置发生冲突，一共有 $m-1$ 种选择。

因此不会导致犯罪的方案是: $m \cdot (m-1)^{n-1}$

则会导致犯罪的方案是： $m^n - m \cdot (m-1)^{n-1}$

代码：

```
 5  typedef long long LL;
 6  const int mod = 100003;
 7
 8  int qmi(int a, LL b) {
 9      int res = 1;
10      while (b) {
11          if (b & 1) res = (LL) res * a % mod;
12          a = (LL) a * a % mod;
13          b >>= 1;
14      }
15      return res;
16  }
18  int main() {
19      int m;
20      LL n;
21      scanf("%d%lld", &m, &n);
22      int res = (qmi(m, n) - (LL)m * qmi(m - 1, n - 1) % mod + mod) % mod;
23      printf("%d\n", res);
24      return 0;
25  }
```

5 [3513] 转圈游戏2 [tg2013]

n个小伙伴（编号从0到n-1）围坐一圈玩游戏。按照顺时针方向给n个位置编号，从0到n-1。最初，第0号小伙伴在第0号位置，第1号小伙伴在第1号位置，...，依此类推。

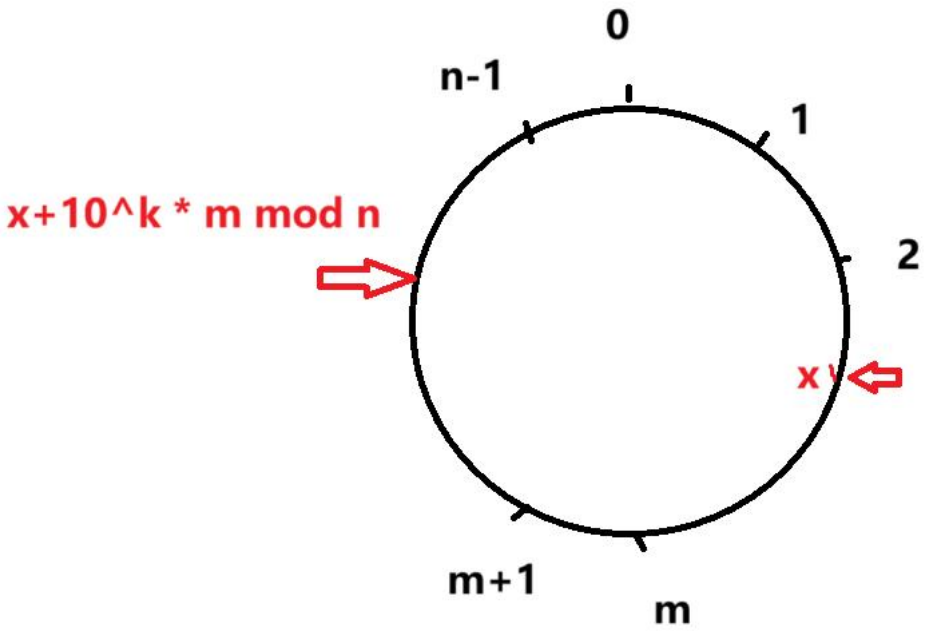
游戏规则如下：每一轮第0号位置上的小伙伴顺时针走到第m号位置，第1号位置小伙伴走到第m+1号位置，...，依此类推，第n-m号位置上的小伙伴走到第0号位置，第n-m+1号位置上的小伙伴走到第1号位置，...，第n-1号位置上的小伙伴顺时针走到第m-1号位置。

现在，一共进行了10^k轮，请问x号小伙伴最后走到了第几号位置。输入4个整数n、m、k、x，，输出 表示10k轮后x号小伙伴所在的位置编号。

$1<n<1,000,000$，$0<m<n$，$0\leqslant x<n$，$0<k<10^9$。

[3513] 转圈游戏2 [快速幂]

输入	输出
10 3 4 5	5



[3513] 转圈游戏2 [快速幂]

```cpp
#include<iostream>
using namespace std;
typedef long long LL;
int qmi(int a,int b,int mod){
    int res=1;//标准的快速幂模板
    while(b){
        if(b&1)
            res=LL(res*a)%mod;
        b>>=1;
        a=LL(a*a)%mod;
    }
    return res;
}
int main(){
    int n,m,k,x;
    scanf("%d%d%d%d",&n,&m,&k,&x);
    printf("%d\n",(x+LL(qmi(10,k,n)*m)%n)%n); //套公式
    return 0;
}
```

矩阵快速幂

快速幂算法不光可以求解普通的快速幂问题，我们还可以用同样的方法对一个任意大小的矩阵求快速幂。快速幂在矩阵乘法中的作用非常的大，普通的两个矩阵相乘的时间复杂度约为 O（ n^3 ）

$$Matrix = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix}^n$$

6 [1945] 斐波那契前 n 项和

大家都知道 Fibonacci 数列吧，

$f_1=1, f_2=1, f_3=2, f_4=3, \cdots, f_n=f_{n-1}+f_{n-2}$。

现在问题很简单，输入 n 和 m，求 fn 的前 n 项和 Sn mod m。

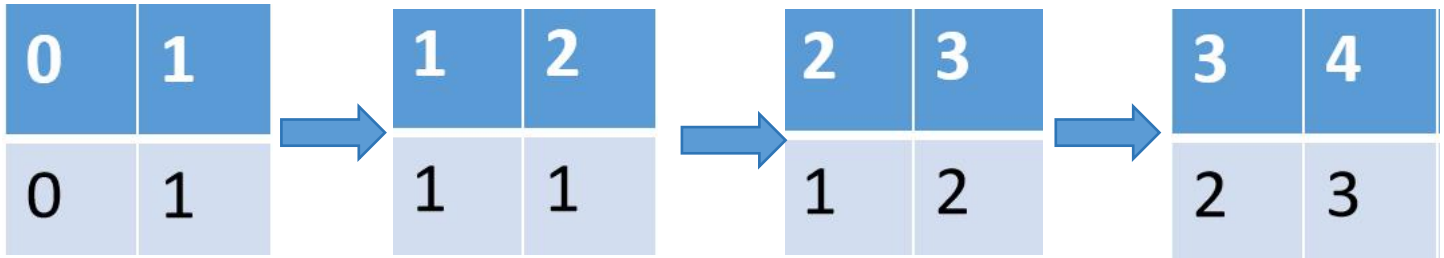
$1 \leqslant n \leqslant 2000000000, 1 \leqslant m \leqslant 1000000010$

输入	输出
5 1000	12

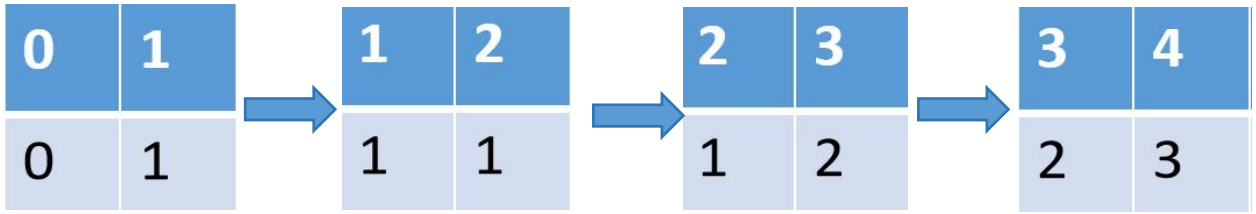
分析：

暴力递归：$O(2^n)$
递推动归：$O(n)$
矩阵快速幂：
$O(\log n)$
数学公示：$O(1)$

0	1	2	3	4	5	6	7
0	1	1	2	3	5	8	13



分析：



$$\left.\begin{matrix}\begin{bmatrix}a & b\\ c & d\end{bmatrix}\begin{pmatrix}1\\0\end{pmatrix}=\begin{pmatrix}1\\1\end{pmatrix}\\ \begin{bmatrix}a & b\\ c & d\end{bmatrix}\begin{pmatrix}1\\1\end{pmatrix}=\begin{pmatrix}2\\1\end{pmatrix}\end{matrix}\right\} \Rightarrow \begin{matrix}a*1+b*0=1\\ c*1+d*0=1\\ a*1+b*1=2\\ c*1+d*1=1\end{matrix} \Rightarrow \begin{matrix}a=1\\ b=1\\ c=1\\ d=0\end{matrix}$$

$$\begin{bmatrix}a & b\\ c & d\end{bmatrix}=\begin{pmatrix}1 & 1\\ 1 & 0\end{pmatrix}$$

$$\begin{bmatrix}f_n\\ f_{n-1}\end{bmatrix}=\begin{bmatrix}1 & 1\\ 1 & 0\end{bmatrix}\begin{bmatrix}f_{n-1}\\ f_{n-2}\end{bmatrix}=\cdots=\begin{bmatrix}1 & 1\\ 1 & 0\end{bmatrix}^{n-2}\begin{bmatrix}f_2\\ f_1\end{bmatrix}$$

分析：

观察原式 $f_n = f_{n-1} + f_{n-2}$

移项可得 $f_{n-2} = f_n - f_{n-1}$

也就是 $f_n = f_{n+2} - f_{n+1}$

将斐波那契的前n项都写成这种形式，得

$$\begin{cases} f_1 = f_3 - f_2 \\ f_2 = f_4 - f_3 \\ f_3 = f_5 - f_4 \\ f_4 = f_6 - f_5 \\ \quad\vdots \\ f_n = f_{n+2} - f_{n+1} \end{cases}$$

分析：

累加所有等式，左边正好是我们要求的答案

而右边，从f_1到f_{n+1}，都互相抵消掉了，得到

$$f_1 + f_2 + \cdots + f_n = f_{n+2} - f_2 = f_{n+2} - 1$$

也就是说，我们就只需要求出 $f_{n+2} - 1$ 即可

$$f_{n+2} + f_{n+1} = f_{n+1} + f_n + f_{n+1}$$

$$\begin{bmatrix} f_{n+2} \\ f_{n+1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} f_{n+1} \\ f_{n+0} \end{bmatrix} = \cdots = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n \begin{bmatrix} f_2 \\ f_1 \end{bmatrix}$$

矩阵快速幂

$$f_{n+2} + f_{n+1} = f_{n+1} + f_n + f_{n+1}$$

$$\begin{bmatrix} f_{n+2} \\ f_{n+1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} f_{n+1} \\ f_{n+0} \end{bmatrix} = \cdots = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n \begin{bmatrix} f_2 \\ f_1 \end{bmatrix}$$

$$[f_{n+2}\ ,\ f_{n+1}] = [f_{n+2}\ ,\ f_{n+1}] \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} = \cdots = [f_2\ ,\ f_1] \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n$$

代码1：

```
5  typedef long long ll;
6  int n, m;
7  ll A[2][2] = {      // 用于快速幂的矩阵
8      {1, 1},
9      {1, 0}
10 };
11 ll res[2] = {1, 0}; // 用于存答案的矩阵（转置）
37 int main()
38 {
39     scanf("%d %d", &n, &m);
40     n += 2; // 求 f[n + 2] 的结果
41     while (n) // 矩阵快速幂板子
42     {
43         if (n & 1) _multi(res, A);
44         multi(A, A);
45         n >>= 1;
46     }
47     printf("%lld\n", res[1] - 1);
48         // 最后 res[1] 即为 f[n + 2] 的结果
49     return 0;
50 }
```

代码2：

```
13  // 计算列矩阵 A 乘方阵 B 的乘积，并存储在 A 中
14  void _multi(ll A[], ll B[][2])
15  {
16      ll ans[2] = {0};
17      for (int i = 0; i < 2; i ++ )
18          for (int j = 0; j < 2; j ++ )
19              ans[i] += A[j] * B[i][j] % m;
20      for (int i = 0; i < 2; i ++ )
21          A[i] = ans[i] % m;
22  }
```

```
41      while (n) // 矩阵快速幂板子
42      {
43          if (n & 1) _multi(res, A);
44          multi(A, A);
45          n >>= 1;
46      }
```

代码3：

```
24  // 计算方阵 A 乘方阵 B 的乘积，并存储在 A 中
25  void multi(ll A[][2], ll B[][2])
26  {
27      ll ans[2][2] = {0};
28      for (int i = 0; i < 2; i ++ )
29          for (int j = 0; j < 2; j ++ )
30              for (int k = 0; k < 2; k ++ )
31                  ans[i][j] += A[i][k] * B[k][j] % m;
32      for (int i = 0; i < 2; i ++ )
33          for (int j = 0; j < 2; j ++ )
34              A[i][j] = ans[i][j] % m;
35  }
```

```
41      while (n) // 矩阵快速幂板子
42      {
43          if (n & 1) _multi(res, A);
44          multi(A, A);
45          n >>= 1;
46      }
```

7 [9256] [lq2015]垒骰子_1700 [dp dfs]

赌圣 atm 晚年迷恋上了垒骰子，就是把骰子一个垒在另一个上边，不能歪歪扭扭，要垒成方柱体。经过长期观察，atm 发现了稳定骰子的奥秘：有些数字的面贴着会互相排斥！我们先来规范一下骰子：1 的对面是 4，2 的对面是 5，3 的对面是 6。

假设有 m 组互斥现象，每组中的那两个数字的面紧贴在一起，骰子就不能稳定的垒起来。atm 想计算一下有多少种不同的可能的垒骰子方式。两种垒骰子方式相同，当且仅当这两种方式中对应高度的骰子的对应数字的朝向都相同。
由于方案数可能过多，请输出模 10^9+7 的结果。。

[9256] [lq2015]垒骰子

第一行两个整数 n，m。$0<n\leqslant 10^9, m\leqslant 36$
n 表示骰子数目。接下来 m 行，每行两个整数 a，b，表示 a 和 b 数字不能紧贴在一起。
输出一行一个数，表示答案模 10^9+7 的结果。

输入	输出
2 1 1 2	544

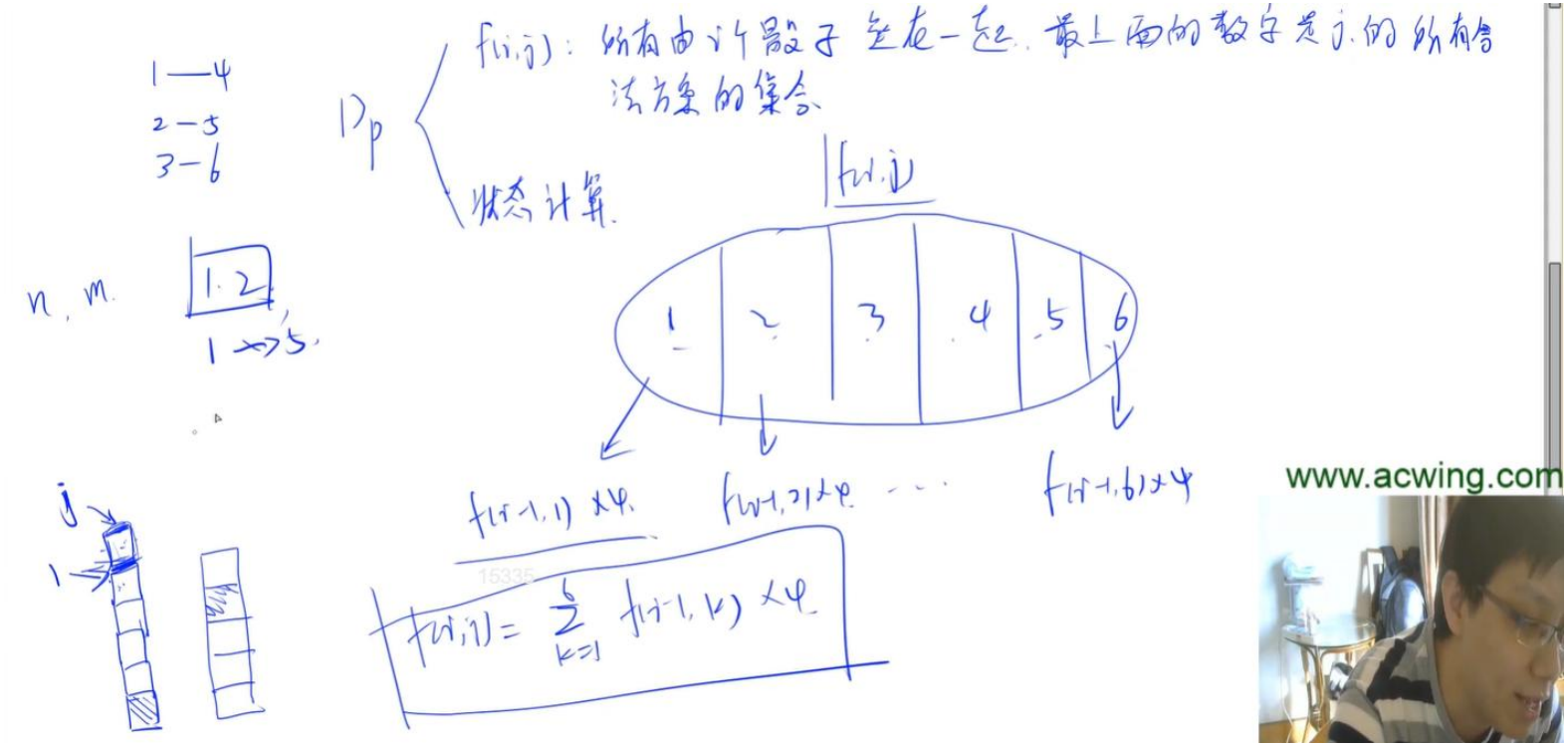
分析：

骰子一个一个往上垒，且求方案数，非常典型的动态规划问题。

1. 求什么？（状态描述）

f[i][j]表示考虑前i个骰子，且第i个骰子上面是点数j时的方案数，这里先没有考虑骰子可以转动。

[9256] [lq2015]垒骰子



[9256] [lq2015]垒骰子

[9256] [lq2015]垒骰子