快速幂、矩阵与线性递推

二进制倍增把 $O(n)$ 次运算压缩到 $O(\log n)$ 次

HGNU-ACM 实验室算法组

零基础精讲：二进制快速幂 + 矩阵乘法结合律 + 3 道经典例题逐行推导

本章目录

二进制拆解与整数快速幂

矩阵乘法与线性递推加速

第 1 节从手算折纸引入，把 $O(b)$ 的连乘压缩到 $O(\log b)$；第 2 节把同一套倍增思想搬到矩阵上，秒杀第 10^{18} 项线性递推。

本章目录（续）

例题精讲一：转圈游戏（一本通 1617）

例题精讲二：矩阵快速幂求斐波那契第 n 项

例题精讲三：迷路（一本通 1647）[提高]

常见错误、检查清单与练习安排

第 3–5 节精讲《转圈游戏》《斐波那契第 n 项》《迷路》三道分层例题；第 6 节汇总高频错误、检查清单与练习安排。

从自动机迈向数学：为什么需要“跳跃”？

- 在上一篇《Trie 树与 AC 自动机》里，我们沿着状态边一步一个字符地往前走。
- 但进入数学专题后，题目常问“操作 10^9 轮”甚至“递推第 10^{18} 项”。
- 如果还老老实实一步一步算，程序还没跑完，比赛早就结束了！
- 先看一个生活场景：一张纸对折 1 次变 2 层，对折 2 次变 4 层，对折 4 次变 16 层，对折 8 次就变 256 层——每次自己乘自己，指数翻倍！

先手算一个小例子

想算 3^8，需要连乘 7 次 3 吗？不需要！只算 3 次平方：$3^2 = 9$，$3^4 = 9^2 = 81$，$3^8 = 81^2 = 6561$。

本页要点

与其一步步乘 b 次，不如让底数不断“自己乘自己”，把指数以 2 的幂次往外跳！

为什么需要快速幂？暴力连乘何时 TLE 与溢出

- 暴力做法：从 1 开始，写一个 `for` 循环把 `res = res * a % mod` 重复执行 b 次。
- 算算操作次数：
  - 若 $b = 10^5$：循环 10^5 次，耗时不到 1 毫秒，暴力能过。
  - 若 $b = 10^9$：循环 10^9 次，C++ 单秒约跑 10^8 次运算，直接 **TLE**（超时）！
  - 若 $b = 10^{18}$：循环 10^{18} 次，需要跑大约 300 年。
- 为什么总要对 *mod* 取模：哪怕 $a = 2, b = 100$，$2^{100} \approx 1.26 \times 10^{30}$，远超 `long long` 上限（约 9.22×10^{18}），不边乘边模就会溢出成负数！

本页要点

当指数 $b \geq 10^8$ 时 $O(b)$ 暴力必 TLE；快速幂只需约 $\log_2 b \leq 60$ 步就能算完 $b = 10^{18}$！

新术语大白话：二进制拆解与两个位运算

- 二进制拆解：把任何正整数写成 $1, 2, 4, 8, 16 \ldots$ 这几个“2 的整次幂积木”相加。例如 $13 = 8 + 4 + 1 = (1101)_2$。
- 取最低位 `b & 1`：看二进制最后一位是不是 1（等价于 `b % 2 == 1`，即当前 b 是不是奇数）。
- 右移一位 `b >>= 1`：把二进制最后一位砍掉（等价于 `b = b / 2`，向下取整）。
- 同余性质（大白话）：乘法过程中随时对 *mod* 取余数，和全部乘完再取余数，结果完全一样：$(x \times y) \bmod m = ((x \bmod m) \times (y \bmod m)) \bmod m$。

本页要点

`b & 1` 负责检查“这一块积木要不要”，`b >>= 1` 负责把指数砍半进入下一块。

写快速幂前，先回答三个问题

把重复动作想清楚再写代码

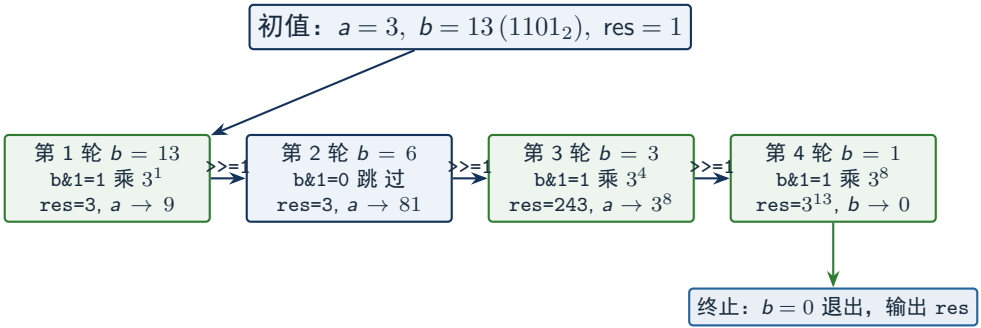
想算 $3^{13} = 3^{8+4+1} = 3^8 \times 3^4 \times 3^1$，每轮循环该做什么？

你要决定的问题	快速幂里怎么选
1. 答案从哪里开始？	乘法的“空箱子”不是 0，而是 `res = 1 % mod`
2. 每轮什么时候把底数乘进答案？	只有当前最低位 `b & 1 == 1` 时，才做 `res = res * a % mod`
3. 每轮底数和指数怎样变化？	不管乘不乘进答案，底数都自乘 `a = a * a % mod`，指数 `b >>= 1`
4. 循环做到什么时候停？	当 `b == 0` 时所有二进制位都处理完了，立刻停止

本页要点

即使某一位是 0 不乘进 `res`，底数 `a` 也必须自乘平方，才能对齐下一位更高的权重！

图解快速幂执行流程：以计算 3^{13} 为例



本页要点

每轮只做一次平方与一次右移；13 有 4 个二进制位，所以只循环 4 轮就结束了！

手算跟踪表：手工追踪 $3^{13} \bmod 100$

轮次	循环前 b	`b > 0?`	`b & 1`	累计答案 `res`	轮末更新后 a 与 b
初值	$13\ (1101_2)$	—	—	$1 \bmod 100 = 1$	$a = 3 \bmod 100 = 3$
第 1 轮	$13\ (1101_2)$	真	1（乘）	$1 \times 3 \bmod 100 = 3$	$a \to 9,\ b \to 6\ (110_2)$
第 2 轮	$6\ (110_2)$	真	0（跳）	3（不变）	$a \to 81,\ b \to 3\ (11_2)$
第 3 轮	$3\ (11_2)$	真	1（乘）	$3 \times 81 \bmod 100 = 43$	$a \to 61,\ b \to 1\ (1_2)$
第 4 轮	$1\ (1_2)$	真	1（乘）	$43 \times 61 \bmod 100 = 23$	$a \to 21,\ b \to 0$
终止检查	0	假	—	保持 **23**	退出循环，返回 23

- 核对真值：$3^{13} = 1594323$，末两位确实是 23！

本页要点

跟踪表一定要写出最后那次“`b > 0` 为假”的终止步，这样才不会搞错退出时的状态。

手算小练习：自己追踪一次 $2^{10} \bmod 1000$

先盖住答案自己算，再核对每一行

$10 = (1010)_2 = 8 + 2$，所以 $2^{10} = 2^8 \times 2^2 = 256 \times 4 = 1024 \equiv 24 \pmod{1000}$。

轮次	当前 b	`b & 1`	累计答案 `res`	轮末 a 与 b
初值	$10\ (1010_2)$	—	1	$a = 2$
第 1 轮	$10\ (1010_2)$	0（跳）	1	$a \to 4,\ b \to 5\ (101_2)$
第 2 轮	$5\ (101_2)$	1（乘）	$1 \times 4 = 4$	$a \to 16,\ b \to 2\ (10_2)$
第 3 轮	$2\ (10_2)$	0（跳）	4	$a \to 256,\ b \to 1\ (1_2)$
第 4 轮	$1\ (1_2)$	1（乘）	$4 \times 256 \bmod 1000 = 24$	$a \to 536,\ b \to 0$
终止	0	假	退出，返回 **24**	—

本页要点

第 4 轮末 a 算完 $256^2 \bmod 1000 = 536$ 后已不再使用，因为 b 已经变成 0。

为什么对？用循环不变量证明快速幂正确

不变量：在每轮开始判断 b > 0 时，恒有 $\text{res} \cdot a^b \equiv a_0^{b_0} \pmod{\text{mod}}$

其中 a_0, b_0 是刚传进函数时的原始底数与原始指数。

阶段	为什么一定成立
初始化	进循环前 $\text{res} = 1 \bmod \text{mod}$，$a = a_0 \bmod \text{mod}$，$b = b_0$，显然 $1 \cdot a_0^{b_0} \equiv a_0^{b_0} \pmod{\text{mod}}$。
保持	若 b 是偶数（$b = 2k$），$\text{res} \cdot a^{2k} = \text{res} \cdot (a^2)^k$，乘积不变；若 b 是奇数（$b = 2k+1$），先把一个 a 乘进 res，剩下 $(\text{res} \cdot a) \cdot (a^2)^k$ 依然等于 $\text{res} \cdot a^{2k+1}$！
终止	循环退出时必有 $b = 0$，此时不变量变成 $\text{res} \cdot a^0 \equiv \text{res} \equiv a_0^{b_0} \pmod{\text{mod}}$。

本页要点

无论奇数还是偶数，每一轮都在“保持 $\text{res} \cdot a^b$ 不变”的前提下把 b 砍半，直到 $b = 0$。

整数快速幂标准模板（11 行）

```
long long qpow(long long a, long long b, long long mod) {
    long long res = 1 % mod;                    // 应对 b = 0 且 mod = 1
    a %= mod;                                   // 进门先取模，防第一步平方溢出
    while (b > 0) {                             // 指数还有二进制位没处理
        if (b & 1) res = res * a % mod;         // 最低位为 1，把当前底数乘进答案
        a = a * a % mod;                        // 底数自乘平方，权重翻倍
        b >>= 1;                                // 指数右移一位（相当于除以 2）
    }
    return res;                                 // 此时 b == 0，res 即为最终余数
}
```

- 为什么全程用 `long long`：只要 $mod \le 3 \times 10^9$，取模后的两数相乘不超过 9×10^{18}，不会溢出。

本页要点

背模板不要只背 `while (b > 0)`，第 2 行的 `1 % mod` 和第 3 行的 `a %= mod` 同样关键！

防坑专题 (1)：为什么初值必须是 `res = 1 % mod`？

- 很多同学习惯顺手写 `long long res = 1;`，平时 99% 的数据都能过，偏偏卡在 `mod = 1`！
- 反例：输入 $a = 5,\ b = 0,\ mod = 1$。
  - 数学上：$5^0 \bmod 1 = 1 \bmod 1 = \mathbf{0}$（任何整数除以 1 的余数都是 0）。
  - 若写 `res = 1;`：因为 $b = 0$，`while (b > 0)` 根本进不去，函数直接返回 **1**，当场 **WA**！
  - 若写 `res = 1 % mod;`：当 $mod = 1$ 时 $1 \bmod 1 = 0$，当 $mod > 1$ 时 $1 \bmod mod = 1$，全部正确！

一句话记住

不要假设 $mod \geq 2$。题目只要写了 $1 \leq m \leq 10^9$，测试点里十有八九藏着 $m = 1$！

本页要点

`res = 1 % mod` 只需要多写 5 个字符，就能彻底消灭 $b = 0, mod = 1$ 的边界 WA。

防坑专题 (2)：进门 a %= mod 与 __int128 防溢出

- 隐患 **A**（底数过大）：若题面给 $a = 10^{12}$, $mod = 1000$，不写 a %= mod 的话，第 1 轮 a * a 就是 10^{24}，远超 long long 上限 9.22×10^{18}，直接溢出变负！
- 隐患 **B**（模数超过 3×10^9）：即使进门做了 a %= mod，若 $mod = 10^{12}$，两个小于 mod 的数相乘也能达到 10^{24}！

```
// 当模数 mod > 3e9（例如 10^12 或 10^18）时，把乘法改成 __int128：
if (b & 1) res = (__int128)res * a % mod;   // 先升到 128 位再取模
a = (__int128)a * a % mod;                  // 最高支持到 1.7e38，不会溢出
```

本页要点

64 位 GCC（NOI Linux / 绝大多数 OJ）都支持 __int128；注意它不能直接 cin/cout，但乘完立刻 % mod 就能安全塞回 long long。

上机验证（1/2）：完整可运行测试代码

```cpp
#include <iostream>
using namespace std;
long long qpow(long long a, long long b, long long mod) {
    long long res = 1 % mod; a %= mod;
    while (b > 0) {
        if (b & 1) res = res * a % mod;
        a = a * a % mod; b >>= 1;
    }
    return res;
}
int main() {
    long long a, b, mod;
    cin >> a >> b >> mod;
    cout << qpow(a, b, mod) << '\n';
    return 0;
}
```

上机验证（2/2）：用 3 组数据亲手测试你的快速幂

三组必测自测样例

- 普通数据：输入 `2 10 1000` $\rightarrow$ 预期输出 `24`。
- 最小边界：输入 `5 0 1` $\rightarrow$ 预期输出 `0`（检验 `res = 1 % mod`）。
- 大底数：输入 `1000000007 3 100` $\rightarrow$ 预期输出 `43`（因为 $7^3 = 343 \equiv 43$）。

本页要点

反事实追问：把上面代码里的 `a %= mod` 删掉，输入 `10000000000 3 100` 会输出什么？（会因为 10^{20} 溢出变负数！）

整数快速幂常见错误对照表

错误写法	为什么错	正确写法与评测结果
`long long res = 1;`	当 $b = 0$ 且 $mod = 1$ 时不进循环，返回 1	`res = 1 % mod;`（避免 **WA**）
漏掉进门 `a %= mod`	当 $a > 3 \times 10^9$ 时首轮 `a * a` 溢出	先写 `a %= mod;`（避免 **WA**）
写成 `res = res * a;`	忘记每步取模，几轮后直接溢出	`res = res * a % mod;`（避免 **WA**）
`mod > 3e9` 仍直接乘	两数相乘超 9.22×10^{18}	强转（`__int128`）再取模（避免 **WA**）
指数 `b` 用 `int` 存	当 $b = 10^{18}$ 时读入就截断出错	`b` 必须开 `long long`（避免 **WA/TLE**）

本页要点

快速幂极少超时，绝大多数失分都来自“忘取模”和“中间乘法溢出”。

延伸了解 (1)：不用 __int128 怎么防 10^{18} 溢出？龟速乘

- 如果模数 $mod \le 10^{18}$ 但评测机禁用了 __int128，可以把 a * b % mod 改成“把 b 按二进制拆开的加法”——龟速乘：

```
long long qmul(long long a, long long b, long long mod) {
    long long res = 0; a %= mod;            // 加法的空箱子是 0
    while (b > 0) {
        if (b & 1) res = (res + a) % mod;   // 只做加法 + 取模，绝不会超 2*mod
        a = (a + a) % mod;                  // 自己加自己，翻倍
        b >>= 1;
    }
    return res;
}
```

本页要点

龟速乘用 $O(\log b)$ 次加法代替一次乘法，只要 $2 \cdot mod$ 不爆 long long 就绝对安全！

延伸了解 (2)：迭代 vs 递归，以及十进制快速幂

- 为什么首选迭代 `while (b > 0)` 而不是递归？
  - 两者都是 $O(\log b)$，但迭代版没有函数调用栈开销，常数更小、代码更短、更不容易写错。
- 如果环境禁用 `__int128` 且 $mod \le 10^{18}$ 怎么办？
  - 把乘法换成“把乘数按二进制拆开的加法”（俗称龟速乘），复杂度变为 $O(\log^2 b)$，虽然慢一点但绝不溢出。
- 如果指数 b 大到连 `long long` 都装不下（如字符串形式的 10^{10000}）？
  - 可以从左到右读十进制每一位做十进制快速幂（每轮让底数做 10 次方），或用欧拉降幂（第 3 章会学）。

本页要点

平时比赛直接用“迭代 + `long long`（模数超 3×10^9 加 `__int128`）”就够打 99% 的快速幂题。

生活引入：怎么把“前两项”打包一起往后推？

- 大家都很熟悉斐波那契数列：$F_1 = 1,\ F_2 = 1,\ F_n = F_{n-1} + F_{n-2}$。
- 平时我们用 `for (int i = 3; i <= n; ++i)` 一项一项往后算，求 F_{100} 只需 98 步。
- 可如果题目要求 $F_{10^{18}} \bmod (10^9 + 7)$，$O(n)$ 循环一定会 **TLE**！
- 换个视角：算下一项只需要最近两项 (F_k, F_{k-1})。如果我们把这两个数打包成一个“状态列向量”，再找一个“规则表”（转移矩阵）去乘它，就能一步从 (F_k, F_{k-1}) 跳到 (F_{k+1}, F_k)！
- 更神奇的是：连乘 $n-2$ 次规则表，就等价于计算这个规则表的 $n-2$ 次方——又能用快速幂了！

本页要点

把递推状态打包成向量，把“向前推一步”写成矩阵乘法，就能用快速幂直接跨过 10^{18} 步！

新术语大白话：矩阵、乘法规则与单位矩阵

- 矩阵（**Matrix**）：就是一张长方形的二维数字表。$d \times d$ 方阵就是 d 行 d 列的二维数组。
- 矩阵乘法 $C = A \times B$：结果第 i 行第 j 列的格子 $C_{i,j}$，等于 A 的第 i 行与 B 的第 j 列对应位置相乘再相加：

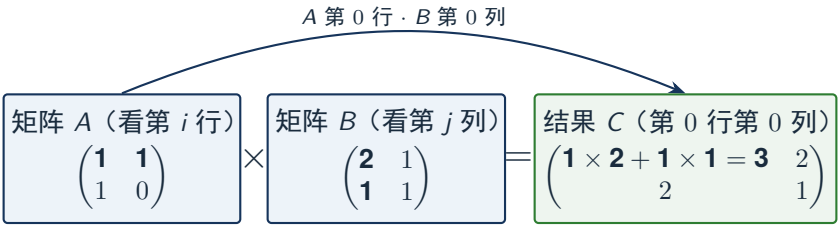
$$C_{i,j} = \sum_{k=0}^{d-1} A_{i,k} \cdot B_{k,j} \pmod{M}$$

- 单位矩阵 I：主对角线（左上到右下）全是 1，其余格子全是 0 的方阵，例如 $I_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$。任何矩阵乘 I 都等于它自己——它就是矩阵世界里的数字 **‘1‘**！

本页要点

单次 $d \times d$ 矩阵乘法需要三重循环，耗时 $O(d^3)$；d 通常很小（$2 \sim 100$）。

图解矩阵乘法：第 i 行乘以第 j 列



手算口诀

算 $C_{i,j}$，左手沿着 A 的第 i 行往右指，右手沿着 B 的第 j 列往下指，两两相乘再求和！

本页要点

注意：A 的列数必须等于 B 的行数才能相乘；方阵（$d \times d$）永远可以自己乘自己。

矩阵乘法的 4 个格子手算跟踪表

计算 $C=\begin{pmatrix}1&1\\1&0\end{pmatrix}\times\begin{pmatrix}2&1\\1&1\end{pmatrix}$ 的全部 4 个元素：

目标位置	取 A 的哪一行	取 B 的哪一列	乘积求和结果
$C_{0,0}$	第 0 行 $(1,1)$	第 0 列 $(2,1)^T$	$1\times2+1\times1=\mathbf{3}$
$C_{0,1}$	第 0 行 $(1,1)$	第 1 列 $(1,1)^T$	$1\times1+1\times1=\mathbf{2}$
$C_{1,0}$	第 1 行 $(1,0)$	第 0 列 $(2,1)^T$	$1\times2+0\times1=\mathbf{2}$
$C_{1,1}$	第 1 行 $(1,0)$	第 1 列 $(1,1)^T$	$1\times1+0\times1=\mathbf{1}$

- 最终得到 $C=\begin{pmatrix}3&2\\2&1\end{pmatrix}$。

本页要点

把 2×2 矩阵乘法的 4 个点积算熟，后面推线性递推转移矩阵就能一眼看穿。

为什么矩阵也能快速幂？结合律成立，但交换律不成立！

- 快速幂唯一要求：运算满足结合律 $(A \cdot B) \cdot C = A \cdot (B \cdot C)$，且有单位元 I。
- 矩阵乘法满足结合律，所以 $T^5 = T \cdot T \cdot T \cdot T \cdot T = T^4 \cdot T^1$，完全能照搬二进制快速幂！
- 重要提醒：矩阵乘法不满足交换律（一般 $A \cdot B \neq B \cdot A$）！

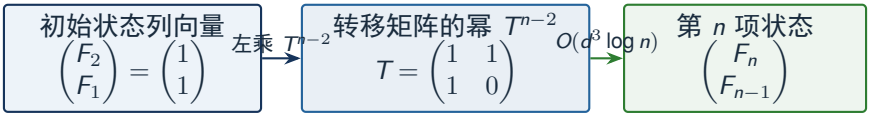
反例：交换律不成立

$\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix}$，而反过来 $\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix}$！

本页要点

因为都是同一个矩阵 T 的幂相乘，$T^a \cdot T^b = T^b \cdot T^a$ 没问题；但不同矩阵左乘/右乘顺序绝不能颠倒！

图解矩阵快速幂：用 T^{n-2} 直接跨到第 n 项



- 原本要从 (F_2, F_1) 一步步推 $n-2$ 次才能到 (F_n, F_{n-1})。
- 现在先用快速幂在 $O(\log n)$ 步内算出 T^{n-2}，再左乘初始向量 $\begin{pmatrix} 1 \\ 1 \end{pmatrix}$，就得到 F_n！

本页要点

矩阵快速幂的总时间复杂度是 $O(d^3 \log n)$：当 $d = 2, n = 10^{18}$ 时，只需做约 $60 \times 8 = 480$ 次整数乘法！

三步列矩阵法 (1)：推导斐波那契转移矩阵

三步法口诀：定向量 $\rightarrow$ 写下步 $\rightarrow$ 抄系数

1. 定状态向量：递推式 $F_{k+1} = F_k + F_{k-1}$ 依赖前两项，将列向量定为 $\begin{pmatrix} F_k \\ F_{k-1} \end{pmatrix}$。
2. 写下一步表达式：把新向量 $\begin{pmatrix} F_{k+1} \\ F_k \end{pmatrix}$ 的每行写成旧向量元素的线性组合：

$$\begin{pmatrix} F_{k+1} \\ F_k \end{pmatrix} = \begin{pmatrix} 1 \cdot F_k + 1 \cdot F_{k-1} \\ 1 \cdot F_k + 0 \cdot F_{k-1} \end{pmatrix}$$

3. 填系数成矩阵：把右边系数按行抄进 2×2 矩阵，即得：

$$\begin{pmatrix} F_{k+1} \\ F_k \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F_k \\ F_{k-1} \end{pmatrix}$$

本页要点

转移矩阵不用死记硬背！只要把新状态用旧状态表示出来，系数直接抄进矩阵行即可。

三步列矩阵法 (2)：带常数项如何升维？

- 假如递推式多了常数项：$F_{k+1} = 2F_k + F_{k-1} + 3$，怎么塞进矩阵里？
- 技巧：在状态向量末尾多加一维常数 1！新向量和旧向量都有一个永远不变的 1：

$$\begin{pmatrix} F_{k+1} \\ F_k \\ 1 \end{pmatrix} = \begin{pmatrix} 2 & 1 & 3 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} F_k \\ F_{k-1} \\ 1 \end{pmatrix}$$

- 检查第三行：$0 \cdot F_k + 0 \cdot F_{k-1} + 1 \cdot 1 = 1$，完美把常数 1 原封不动传到下一轮！

本页要点

遇到 $+c$ 常数项就加一维 1；遇到 $+k$ 等一次项，同样可以多加一维把 k 和 1 一起带着走！

三步列矩阵法 (3)：求前缀和 $S_n = F_1 + F_2 + \cdots + F_n$

- 如果题目不求 F_n，而是求前 n 项和 $S_n = S_{n-1} + F_n$ 怎么办？
- 把 S_k 也塞进状态向量！设状态向量为 $\begin{pmatrix} S_k & F_{k+1} & F_k \end{pmatrix}^T$：

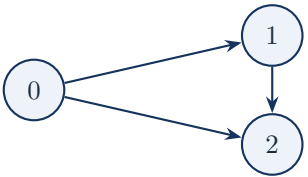
$$\begin{pmatrix} S_{k+1} \\ F_{k+2} \\ F_{k+1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} S_k \\ F_{k+1} \\ F_k \end{pmatrix}$$

- 第 1 行：$S_{k+1} = 1 \cdot S_k + 1 \cdot F_{k+1} + 0 \cdot F_k$；第 $2, 3$ 行照搬斐波那契转移。

本页要点

递推里需要累加什么量，就把这个量加入状态向量；矩阵维数只增加 1，依然是 $O(d^3 \log n)$！

图论延伸：邻接矩阵幂 $A^k[u][v]$ 的路径计数



$A[u][v] = 1$ 表示 $u \to v$ 有 1 条边
$A^2[0][2] = \sum_w A[0][w] \cdot A[w][2]$
$= A[0][1]A[1][2] = 1$（恰好 2 步：$0 \to 1 \to 2$）

- 通用结论：在有向图（边权均为 1）中，邻接矩阵 A 的 k 次幂 $A^k[u][v]$ 恰好等于从点 u 出发、走恰好 k 步到达点 v 的不同路径总数！

本页要点

“走恰好 k 步”的路径计数与“线性递推第 k 项”底层完全一致，共用同一套矩阵快速幂模板。

图论邻接矩阵幂手算跟踪表：3 个点走 $k=2$ 步

对上一页的 3 点有向图（边：$0\to1,\ 1\to2,\ 0\to2$），写下邻接矩阵 A 并手算 A^2：

矩阵	具体数值	含义解释
A	$\begin{pmatrix}0&1&1\\0&0&1\\0&0&0\end{pmatrix}$	走恰好 1 步的路径数（如 $A[0][1]=1$）
A^2	$\begin{pmatrix}0&0&1\\0&0&0\\0&0&0\end{pmatrix}$	走恰好 2 步的路径数（仅 $A^2[0][2]=1$，即 $0\to1\to2$）

本页要点

只有当 u 到 w 有边且 w 到 v 有边时，$A[u][w]\cdot A[w][v]=1$ 才会对 $A^2[u][v]$ 贡献 1 条两步路径！

矩阵快速幂模板代码（1/2）：定义与矩阵乘法

```cpp
using Mat = array<array<long long, 2>, 2>;  // 2x2 矩阵，可用 {} 默认全零初始化
const long long MOD = 1000000007;

Mat mul(const Mat &A, const Mat &B) {
    Mat C{};                                  // 局部矩阵必须清零！
    for (int i = 0; i < 2; ++i)
        for (int k = 0; k < 2; ++k) if (A[i][k]) // i-k-j 顺序更快且跳过 0 项
            for (int j = 0; j < 2; ++j)
                C[i][j] = (C[i][j] + A[i][k] * B[k][j]) % MOD;
    return C;
}
```

本页要点

写 `i-k-j` 循环顺序对 CPU 缓存更友好；加上 `if (A[i][k])` 还能在稀疏矩阵时大幅提速。

矩阵快速幂模板代码（2/2）：幂次函数

```
Mat mat_pow(Mat base, long long e) {
    Mat ans{{{1, 0}, {0, 1}}};              // 初始化为单位矩阵 I（矩阵的 "1"）
    while (e > 0) {
        if (e & 1) ans = mul(ans, base);    // 当前二进制位为 1，乘入结果矩阵
        base = mul(base, base);             // 转移矩阵自乘平方
        e >>= 1;                            // 指数右移一位
    }
    return ans;                             // 若 e == 0，直接返回单位矩阵 I
}
```

- 为什么 ans 必须设成单位矩阵 /：就像整数快速幂 res 要设成 1 一样，全 0 矩阵乘任何矩阵都还是全 0！

本页要点

把整数快速幂里的 1 换成 I、把 * 换成 mul(A, B)，就是全部的矩阵快速幂！

通用 $d \times d$ 矩阵怎么封装？（用 struct Mat）

如果 d 不是固定的 2（例如 $d = 3$ 或 $d = 90$），可以用结构体封装一个通用矩阵：

```
struct Mat {
    int d;
    vector<vector<long long>> a;
    Mat(int n, long long diag = 0) : d(n), a(n, vector<long long>(n, 0)) {
        for (int i = 0; i < d; ++i) a[i][i] = diag; // diag=1 就是单位阵 I
    }
};
```

- `Mat I(d, 1);` 一行代码就能造出任意大小的单位矩阵；`Mat C(d);` 默认全零！

本页要点

用 struct Mat 配合构造参数 diag，以后写 3×3、90×90 矩阵再也不用手动写大括号了。

为什么对？矩阵快速幂的循环不变量

阶段	为什么一定成立（不变量：$\text{ans}\cdot\text{base}^{e}\equiv T^{e_0}\pmod M$）
初始化	循环前 $\text{ans}=I$，$\text{base}=T$，$e=e_0$。因为 $I\cdot T^{e_0}=T^{e_0}$，成立。
保持	因矩阵乘法满足结合律：若 e 偶则 $\text{ans}\cdot(\text{base}^2)^{e/2}$ 不变；若 e 奇则 $(\text{ans}\cdot\text{base})\cdot(\text{base}^2)^{(e-1)/2}$ 仍等于 $\text{ans}\cdot\text{base}^{e}$。
终止	当 $e=0$ 退出循环时，$\text{base}^0=I$，因此 $\text{ans}=\text{ans}\cdot I\equiv T^{e_0}\pmod M$。

本页要点

唯一需要矩阵乘法满足的性质就是“结合律”，这正是快速幂能从整数无缝推广到矩阵的根源。

上机验证与反事实追问

调用模板求斐波那契第 n 项（$n \le 2$ 特判输出 1，否则输出 `(ans[0][0] + ans[0][1]) % MOD`）：

- 最小边界：输入 `1` 或 `2` $\to$ 预期输出 `1`。
- 普通数据：输入 `5` $\to$ 预期输出 `5`（因为 $1, 1, 2, 3, 5$）。
- 大数据取模：输入 `50` $\to$ 预期输出 `586268941`（$F_{50} \equiv 586268941 \pmod{10^9+7}$）。

反事实追问

1. 若把 `Mat ans{{{1, 0}, {0, 1}}}` 错写成 `Mat ans{}`，任何 $n \ge 3$ 会输出什么？（全 0！）
2. 若忘记特判 $n \le 2$，输入 $n = 1$ 时 `e=-1` 不进循环凑巧对，但算 $n = 0$ 会出错，故必须显式特判！

本页要点

写完矩阵快速幂后，务必用 $n = 1, 2, 3, 4, 5$ 全部跑一遍核对前五项！

矩阵快速幂常见错误对照表

错误写法	为什么错	正确写法与评测结果
`Mat ans{};`（全零矩阵）	零矩阵乘任何矩阵都是零	主对角线设为 1 的单位矩阵 I（避免 **WA**）
`Mat C;` 局部不初始化	栈上局部数组残留垃圾值	写 `Mat C{};` 清零（避免 **WA**）
递推幂次写错（多乘一次）	已有 F_1, F_2 却算了 T^n	从第 2 项到第 n 项是 T^{n-2}（避免 **WA**）
累加时没取模直接到最后模	d 项累加若每项可达 M^2 会溢出	每次累加都 `% MOD`（避免 **WA**）
不该用矩阵时强行用	$d = 500$ 时 $d^3 \log n > 10^{10}$	d 太大时改用其他算法（避免 **TLE**）

本页要点

矩阵快速幂只在 d 较小（通常 $d \le 100$）而 n 极大（如 $10^9 \sim 10^{18}$）时才是神兵利器。

例题 1　先读懂要求：转圈游戏（一本通 1617）[基础]

大白话题意

n 人围成一圈（顺时针编号 $0 \sim n-1$），最初第 x 号在位置 x。每轮所有人顺时针走 m 步（过 $n-1$ 回 0）。共进行 10^k 轮，求第 x 号最终位置编号。

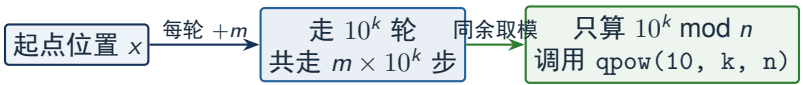
项目	具体要求
输入	一行 4 个整数 n, m, k, x（$1 < n < 10^6,\ 0 < m < n,\ 0 \le x < n,\ 0 < k < 10^9$）
处理	计算经过 10^k 轮后的最终位置编号
输出	一行 1 个整数，表示最终所在的位置编号
原题样例	输入 `10 3 4 5` $\rightarrow$ 输出 `5`

本页要点

注意 k 最大达到 10^9，总轮数 10^k 是一个有一亿个零的超级大数，绝对不能算出来！

例题 1　怎么想到的：把 10^k 轮压缩成一次跳跃

- 暴力思路：写一个循环跑 10^k 轮，每轮 `pos = (pos + m) % n`。但 10^k 太大，连 $k = 9$ 都要跑 10^9 轮（TLE），更别说 $k = 10^9$！
- 发现性质：每一轮都是固定往前走 m 步，走 10^k 轮相当于总共往前走了 $m \times 10^k$ 步！
- 同余化简：最终位置就是 $(x + m \times 10^k) \bmod n = (x + (m \bmod n) \times (10^k \bmod n)) \bmod n$。



本页要点

用同余性质把 10^k 拆出来单独对 n 取模，就变成了标准的整数快速幂！

例题 1　同余性质大白话：为什么可以先模 n 再乘？

- 在一圈 n 个格子里，每走满 n 步就会回到原点，相当于“白走一圈”。
- 所以总步数 $m \times 10^k$ 里面，所有整圈（n 的倍数）都可以提前扔掉！
- 数学表达：

$$(x + m \times 10^k) \bmod n = \big(x + (m \bmod n) \times (10^k \bmod n)\big) \bmod n$$

- 这样一来，`m % n` 不超过 n，`qpow(10, k, n)` 也不超过 n，两者相乘最大不超过 $n^2 \le 10^{12}$，在 `long long` 内绝对安全！

本页要点

取模就像“绕操场跑圈只看最后停在跑道第几米”，整圈数可以随时丢掉不影响结果。

例题 1　样例手推跟踪表

先手推原题样例 `n=10`，`m=3`，`k=4`，`x=5`，再追一组不整除的小样例 `n=7`，`m=3`，`k=2`，`x=1`：

样例	第 1 步：算 $10^k \bmod n$	第 2 步：算总位移	第 3 步：加起点模 n	输出
`10 3 4 5`	$10^4 \bmod 10 = 0$	$3 \times 0 = 0$	$(5+0) \bmod 10 = \mathbf{5}$	5
`7 3 2 1`	$10^2 \bmod 7 = 100 \bmod 7 = 2$	$3 \times 2 = 6$	$(1+6) \bmod 7 = \mathbf{0}$	0

- 检验 `7 3 2 1`：走 $10^2 = 100$ 轮，每轮 3 步共 300 步；从 1 号出发，$1 + 300 = 301 = 7 \times 43$，确实停在 0 号位置！

本页要点

用“手算小样例（$k = 2$）”和“原题整除样例”双保险检验公式，确认加减方向和取模无误。

例题 1　核心步骤表与相对模板的变化

按顺序计算	纸上得到	核对问题
1. 读入 n, m, k, x	确定圈长 n、步长 m	n, m, k, x 均用 `long long` 读入吗？
2. 调用 `qpow(10, k, n)`	$10^k \bmod n$ 的余数	进循环前做 `a %= mod` 了吗？
3. 计算最终位置	$(x + m \bmod n \cdot \mathrm{pw}) \bmod n$	最外层是否再套了一次 `% n`？

相对整数快速幂模板改了哪里？

`qpow` 函数一字未改；只是把主函数里的底数固定为 10、指数为 k、模数为 n，最后再做一步线性偏移 $(x + m \cdot 10^k) \bmod n$。

本页要点

遇到“每轮走固定步长、共走 10^k 轮”的模型，直接把步数合并成乘法再用快速幂。

例题 1　核心代码（1/2）：快速幂函数（code/pow_circle.cpp）

```cpp
#include <bits/stdc++.h>
using namespace std;

// 一本通 1617 转圈游戏：快速幂算 10^k mod n
long long qpow(long long a, long long b, long long mod) {
    long long res = 1 % mod; a %= mod;      // 边界防坑：mod=1 与大底数
    while (b > 0) {
        if (b & 1) res = res * a % mod;     // 二进制位为 1 则乘入答案
        a = a * a % mod;
        b >>= 1;
    }
    return res;
}
```

本页要点

这里 qpow 与第 1 节模板完全一致，可直接作为比赛时的标准子函数复用。

例题 1　核心代码（2/2）：主函数计算位置（code/pow_circle.cpp）

```
int main() {
    ios::sync_with_stdio(false); cin.tie(nullptr);
    long long n, m, k, x;
    if (cin >> n >> m >> k >> x)
        cout << (x + m % n * qpow(10, k, n)) % n << '\n';
    return 0;
}
```

- 运算顺序：`m % n * qpow(10, k, n)` 在 C++ 中从左到右结合，等价于 `(m % n) * qpow(10, k, n)`。
- 为何外层还要 `% n`：因为 $x+$ 偏移量 可能大于等于 n，需要绕回 $0 \sim n-1$。

本页要点

把 `m % n` 写在乘号左边，就算 m 未来加强到 10^{18} 也不会在相乘时溢出。

例题 1　易错点与 3 组自测边界

- 易错 **1**：用 `pow(10, k)` 浮点函数！`pow` 返回 `double`，精度只有 15 位且 $k = 10^9$ 时直接溢出成 `inf`。
- 易错 **2**：变量全开 `int`！若 $n = 10^6$，`m * qpow(10, k, n)` 最高可达 10^{12}，在 `int` 里直接溢出！

自测类型	输入数据	预期输出与验证要点
原题样例	`10 3 4 5`	`5`（$10^4 \bmod 10 = 0$，回到 5）
最小边界	`2 1 0 1`	`0`（$10^0 = 1$ 轮，$(1+1) \bmod 2 = 0$）
大数边界	`999999 999998 999999999 0`	检查不溢出 `int`、不超时

本页要点

C++ 算法竞赛里凡是整数求幂取模，一律禁止使用 `<cmath>` 里的 `pow()`！

例题 2　先读懂要求：斐波那契第 n 项（模 10^9+7）[基础]

大白话题意

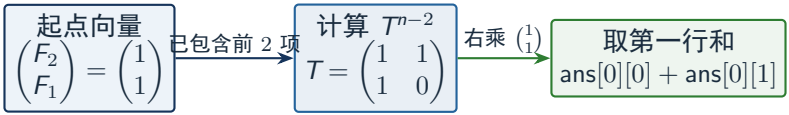
已知斐波那契数列 $F_1=1,\ F_2=1$，从第 3 项起满足 $F_n=F_{n-1}+F_{n-2}$。给定正整数 n（可达 10^{18}），求 $F_n \bmod (10^9+7)$。（注：一本通 1642 则改为输入 n, m 求 $F_n \bmod m$，做法完全相同。）

项目	具体要求
输入	一行 1 个正整数 n（$1 \le n \le 10^{18}$）
处理	用 2×2 矩阵快速幂计算 $F_n \bmod 1000000007$
输出	一行 1 个整数，表示第 n 项取模后的结果
验证样例	输入 5 $\to$ 输出 5；输入 1 $\to$ 输出 1

本页要点

$n \le 10^{18}$ 是矩阵快速幂的经典信号：线性递推 + 超大项数 = 矩阵加速！

例题 2　怎么想到的：从 T^{n-2} 左乘 $(F_2, F_1)^T$



- 为什么是 `ans[0][0] + ans[0][1]`？把结果矩阵 $\text{ans} = T^{n-2} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ 乘以初始列向量 $\begin{pmatrix} 1 \\ 1 \end{pmatrix}$：

$$\begin{pmatrix} F_n \\ F_{n-1} \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} a+b \\ c+d \end{pmatrix}$$

第一行就是 $F_n = a + b = \text{ans}[0][0] + \text{ans}[0][1]$！

本页要点

因为初始列向量恰好是全 1 的 $(1,1)^T$，所以连向量乘法都省了，直接把第一行两数相加即可！

例题 2　样例手推跟踪表：手推 $n = 5$

当 $n = 5$ 时，幂次 $e = n - 2 = 3 = (11)_2$，初始 ans $= I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$，base $= \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$：

轮次	e	e & 1	更新后 ans	轮末 base 与 e
初值	$3\ (11_2)$	—	$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$	base $= \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$
第 1 轮	$3\ (11_2)$	1（乘）	ans $\times$ base $= \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$	base $\to \begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix}$，$e \to 1$
第 2 轮	$1\ (1_2)$	1（乘）	$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 3 & 2 \\ 2 & 1 \end{pmatrix}$	base $\to \begin{pmatrix} 5 & 3 \\ 3 & 2 \end{pmatrix}$，$e \to 0$
终止	0	假	退出循环	输出 $3 + 2 = \mathbf{5}$

本页要点

第 2 轮退出后，ans$[0][0] +$ ans$[0][1] = 3 + 2 = 5$，与 $F_5 = 5$ 分毫不差！

例题 2　核心步骤表与相对模板的变化

按顺序计算	纸上得到	核对问题
1. 特判 $n \le 2$	直接输出 1（若模 $m = 1$ 需模 m）	$n = 1$ 时防止 $e = n - 2 < 0$ 吗？
2. 初始化 ans 和 base	ans 为 I，base 为 $\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$	ans 确定没写成全零矩阵？
3. 对 $e = n - 2$ 做快速幂	得到 T^{n-2}	每次矩阵乘法都有 % MOD 吗？
4. 输出第一行之和	(ans[0][0] + ans[0][1]) % MOD	加完之后再模一次 MOD 吗？

如果要做一本通 1642（输入 n, m 求 $F_n \bmod m$）怎么改？

把全局常量 MOD 改成读入的变量 m，并把 $n \le 2$ 时的输出改成 1 % m 即可！

本页要点

改任意模数时千万别忘了 $n \le 2$ 特判也要输出 1 % m，防 $m = 1$ 的边界坑！

例题 2　延伸：怎么改成任意模数 m（一本通 1642）？

一本通 1642《Fibonacci 第 n 项》给定 n, m（$1 \le n \le 2 \times 10^9,\ 1 \le m \le 10^9 + 10$），只需改 3 处：

修改位置	原代码（固定模 10^9+7）	改写后（任意模数 m）
模数变量	`const long long MOD = 1e9+7;`	读入 `long long n, m;` 传入 `mul`
$n \le 2$ 特判	`if (n <= 2) { cout << 1; }`	`if (n <= 2) { cout << 1 % m; }`
最终输出	`(ans[0][0]+ans[0][1]) % MOD`	`(ans[0][0]+ans[0][1]) % m`

本页要点

切记 $n \le 2$ 特判也要写 `1 % m`，否则遇到测试点 $n = 1, m = 1$ 会输出 1 而不是 0！

例题 2　核心代码（1/2）：矩阵定义与乘法（code/mat_pow_fib.cpp）

```cpp
#include <bits/stdc++.h>
using namespace std;
using Mat = array<array<long long, 2>, 2>;
const long long MOD = 1000000007;

Mat mul(const Mat &A, const Mat &B) {
    Mat C{};
    for (int i = 0; i < 2; ++i)
        for (int k = 0; k < 2; ++k) if (A[i][k])
            for (int j = 0; j < 2; ++j)
                C[i][j] = (C[i][j] + A[i][k] * B[k][j]) % MOD;
    return C;
}
```

本页要点

与模板一致：`Mat C{}` 保证局部矩阵全零，`i-k-j` 循环顺序提升缓存命中率。

例题 2　核心代码（2/2）：主函数快速幂（code/mat_pow_fib.cpp）

```cpp
int main() {
    ios::sync_with_stdio(false); cin.tie(nullptr);
    long long n; if (!(cin >> n)) return 0;
    if (n <= 2) { cout << 1 << '\n'; return 0; }
    Mat ans{{{1, 0}, {0, 1}}}, base{{{1, 1}, {1, 0}}}; // 单位矩阵 I
    long long e = n - 2;
    while (e > 0) {
        if (e & 1) ans = mul(ans, base);
        base = mul(base, base);
        e >>= 1;
    }
    cout << (ans[0][0] + ans[0][1]) % MOD << '\n';
    return 0;
}
```

本页要点

记住 `e = n - 2` 而不是 `n`，因为初始向量已经握着 F_1 和 F_2 了！

例题 2　易错点与 3 组自测边界

- 易错 **1**：n 开成 int！题目 $n \le 10^{18}$，读入 int 直接溢出。
- 易错 **2**：把 (ans[0][0] + ans[0][1]) % MOD 漏了外层取模，两个接近 10^9 的数相加会超 10^9+7。

自测类型	输入数据	预期输出与验证要点
最小边界	1 和 2	均输出 1（检验特判）
小样例	5	输出 5（检验 $e = n-2 = 3$ 正确）
极大边界	1000000000000000000	输出 209783453（检验 long long 与复杂度）

本页要点

凡是递推求第 n 项，交题前必测 $n = 1, 2, 3, 4, 5$ 五个数，一秒钟就能排除偏移量错误。

例题 3　先读懂要求：迷路（一本通 1647 / SCOI 2009）[提高]

大白话题意

给定 N 个点的有向图（编号 $0 \sim N-1$）。邻接矩阵第 i 行 j 列为 0 表示无边，为 $1 \sim 9$ 表示从 i 到 j 耗时 $1 \sim 9$。从点 0 出发、不能原地停留，求恰好在 T 时刻到达点 $N-1$ 的路径数 $\bmod 2009$。

项目	具体要求
输入	首行 N, T（$2 \le N \le 10,\ 1 \le T \le 10^9$）；下接 N 行长度为 N 的数字串
处理	求从 0 到 $N-1$ 总耗时恰好为 T 的有向路径数 $\bmod 2009$
输出	一个整数，表示方案数 $\bmod 2009$
原题样例	输入 `2 2 / 11 / 00` $\rightarrow$ 输出 `1`（路径 $0 \to 0 \to 1$）

本页要点

边权不是 1（而是 $1 \sim 9$），但 T 高达 10^9 且 N 极小（≤ 10），这是拆点 + 矩阵快速幂的绝佳信号！

例题 3　怎么想到的：把长度 w 的边拆成 w 步“时间点”

- 我们知道：邻接矩阵 $A^T[0][N-1]$ 能算“走恰好 T 步”的方案数，但前提是每条边耗时必须都是 1！
- 这题边权 $w \in [1,9]$，怎么化为 1？拆点！
- 把每个点 u 拆成 9 个状态点 $(u,0),(u,1),\ldots,(u,8)$，其中 $(u,0)$ 表示“刚到达 u”，(u,t) 表示“走一条边还需要 t 秒才能到达 u”：



- 内部连边：$(v,t) \to (v,t-1)$（$t = 1 \ldots 8$）；原图边 $u \xrightarrow{w} v$：从 $(u,0)$ 连向 $(v,w-1)$！

本页要点

总点数变成 $9N \le 90$，所有边权全变成 1！做一次 90×90 矩阵快速幂 A^T，复杂度约 $90^3 \log T \approx 2.2 \times 10^7$，轻松过关！

例题 3　样例手推跟踪表

以原题样例 1（$N=2, T=2$，邻接表：$0\to 0$ 耗时 1，$0\to 1$ 耗时 1）为例：

步骤	转移含义	状态与结果
建边（$w=1$）	$u \xrightarrow{1} v$ 连 $(u,0)\to(v,0)$	$(0,0)\to(0,0)$ 且 $(0,0)\to(1,0)$
时刻 $t=1$	从 $(0,0)$ 走 1 步	可到达 $(0,0)$（1 种）和 $(1,0)$（1 种）
时刻 $t=2$	再走 1 步（因点 1 无出边）	只能从 $(0,0)$ 走到 $(0,0)$ 或 $(1,0)$
查询答案	取 A^2 中起点 $(0,0)$ 到终点 $(1,0)$	恰好 1 种方案（$0\to 0\to 1$），输出 1

本页要点

注意终点是 $(N-1,0)$ 而不是 $(N-1,t)$，因为只有走到 0 层才算真正到达了点 $N-1$！

例题 3　核心步骤表与相对模板的变化

按顺序计算	纸上得到	核对问题
1. 计算总点数 $S = 9N$	编号函数 `id(u, t) = u*9 + t`	是否覆盖了 $t = 0 \sim 8$ 共 9 层？
2. 连内部倒计时边	$(u, t) \to (u, t-1)$ 设为 1	从 $t = 1$ 到 8，不要连出界！
3. 读入字符矩阵建边	若 $w > 0$，连 $(u, 0) \to (v, w-1)$	字符 `'0'..'9'` 转数字记得减 `'0'` 吗？
4. 算 A^T 并输出	`ans[id(0,0)][id(N-1,0)]`	模数是 2009，别写成 `1e9+7`！

相对斐波那契矩阵模板改了哪里？

矩阵维数从固定 2×2 变成动态 $S \times S$（最大 90×90）；模数从 $10^9 + 7$ 改成 2009；矩阵乘法加上 `if (A[i][k])` 剪枝后能提速数倍！

本页要点

“边权很小（≤ 9）但步数/时间极长（$\leq 10^9$）”是拆点化边权为 1 的标准特征。

例题 3　复杂度精算：为什么 90×90 矩阵不会 TLE？

- 拆点后总点数 $S = 9N \le 90$，单次矩阵乘法理论最坏运算次数为 $90^3 = 729,000$ 次。
- 当 $T = 10^9$ 时，快速幂最多做 $\lfloor \log_2(10^9) \rfloor + 1 = 30$ 次矩阵乘法（实际平均 15 次 `ans*base` + 30 次 `base*base`）。
- 总乘加次数约 $45 \times 7.29 \times 10^5 \approx 3.28 \times 10^7$ 次，在 1 秒内可以轻松跑完！
- 再加上 `if (A[i][k] == 0) continue;` 剪枝，前几轮矩阵极其稀疏，实际耗时通常不到 50 毫秒。

本页要点

先算复杂度再写代码：$90^3 \times 45 \approx 3.3 \times 10^7 < 10^8$，心里有底才敢写！

例题 3　核心代码（1/2）：通用矩阵乘法与快速幂

```cpp
const int MOD = 2009;
using Mat = vector<vector<int>>;
Mat mul(const Mat &A, const Mat &B, int S) {
    Mat C(S, vector<int>(S, 0));
    for (int i = 0; i < S; ++i)
        for (int k = 0; k < S; ++k) if (A[i][k]) // 稀疏剪枝，提速关键
            for (int j = 0; j < S; ++j)
                C[i][j] = (C[i][j] + A[i][k] * B[k][j]) % MOD;
    return C;
}
```

本页要点

因为模数只有 2009，两个数相乘最大 $2008^2 \approx 4 \times 10^6$，用 int 就能存下且更快！

例题 3　核心代码（2/2）：拆点建图与矩阵求幂

```
int id(int u, int t) { return u * 9 + t; }    // 点 u 的第 t 层编号 (0<=t<9)
// 建图：在 S = 9 * N 的矩阵 A 中初始化边
for (int u = 0; u < N; ++u) {
    for (int t = 1; t < 9; ++t) A[id(u, t)][id(u, t - 1)] = 1; // 倒计时 1 秒
    string s; cin >> s;
    for (int v = 0; v < N; ++v) {
        int w = s[v] - '0';
        if (w > 0) A[id(u, 0)][id(v, w - 1)] = 1; // u->v 耗时 w
    }
}
Mat res = mat_pow(A, T, 9 * N);
cout << res[id(0, 0)][id(N - 1, 0)] << '\n';
```

本页要点

因为 $w \geq 1$，所以 $w-1 \in [0,8]$ 恰好落在 9 层之内，再走 $w-1$ 步倒计时就正好到 $(v,0)$！

例题 3　易错点与 3 组自测边界

- 易错 **1**：输入是紧挨着的数字串（如 `12045`），没有空格，不能直接 `cin >> int`，必须按字符或字符串读！
- 易错 **2**：忘了 `if (A[i][k])` 跳过 0 项！$90^3 \times 30 \approx 2.2 \times 10^7$ 次取模如果全做，常数偏大容易卡常。

自测类型	输入数据	预期输出与验证要点
原题样例 1	`2 2 / 11 / 00`	输出 `1`
原题样例 2	`5 30 / 12045 ...`	输出 `852`（检验 $w > 1$ 的拆点连边与取模 2009）
不可达边界	`2 5 / 00 / 00`	输出 `0`（检验矩阵全零时答案正确）

本页要点

矩阵乘法里加一句 `if (A[i][k] == 0) continue;`，是稀疏矩阵快速幂最廉价也最有效的常数优化！

本章常见错误对照表

题号 / 场景	常见错误写法	正确做法与原因
1616 / 1617 快速幂	`res = 1;` 不对 `mod` 取模	必须写 `res = 1 % mod;`，否则 $b = 0, mod = 1$ 时 WA
1616 / 1617 快速幂	没写 `a %= mod;` 直接平方	底数较大时第 1 轮 `a * a` 直接溢出变负
1642 斐波那契	`Mat ans{};` 全零初值	必须设为主对角线为 1 的单位矩阵 I
1642 斐波那契	幂次写成 T^n 而不是 T^{n-2}	已有 F_1, F_2，多乘 2 次导致答案错位，且需特判 $n \le 2$
1647 迷路	把输入行当整数读入	行内无空格，必须用 `string` 逐字符 `s[v]-'0'` 读入

本页要点

“底数进门先取模，`res` 初值 `1 % mod`，矩阵初值是单位阵，乘方次数数清楚！”

按评测结果快速定位 Bug（CE / WA / RE / TLE）

评测结果	本章最常见原因	第一时间检查哪里
CE	`array` 未引头文件或 `__int128` 误用于本地 32 位编译器	加 `#include <bits/stdc++.h>`；确认本地编译开启 64 位
WA	输出全 0 / 负数 / 差一位	全 0 查单位矩阵；负数查乘法先溢出；差一位查 $n-2$ 与 $n \le 2$ 特判
RE	拆点后矩阵下标越界（如 $u \times 9 + t$ 超出数组大小）	确认数组至少开到 $9N$（如 95×95），且 $w-1 \in [0,8]$
TLE	没用快速幂（$O(b)$ 硬乘）或 90×90 矩阵乘法未剪枝	检查循环是否写对 `b >>= 1`；在矩阵乘法内层前加 `if (A[i][k])`

本页要点

算出来全 0 查单位阵，突然变负查溢出，差一项查幂次偏移，超时查 `b >>= 1` 和稀疏剪枝！

提交前检查清单（六必查）

1. 快速幂函数的返回初值是否写成了 `res = 1 % mod`，底数进门前是否先做了 `a %= mod`？
2. 如果模数超过了 3×10^9，单次乘法是否使用了 `__int128` 防止中间乘法溢出？
3. 矩阵乘法中新建结果矩阵 `C` 时是否全部清零初始化（如 `Mat C{}`），循环中是否每步取模？
4. 矩阵快速幂的累乘答案矩阵 `ans` 是否正确初始化为单位矩阵 I（主对角线为 1）？
5. 线性递推求第 n 项时，是否手动校验了 $n = 1, 2, 3$ 的边界情况与幂次偏移量（如 $n-2$）？
6. 遇到图论拆点或无空格字符矩阵时，是否按字符串读入且数组空间开够 $9N$？

最后一步

用手算小样例：$3^0 \bmod 1 = 0$、斐波那契 $n = 1, 2, 3, 4, 5$ 依次输出 $1, 1, 2, 3, 5$ 验证后再提交！

本章全部练习与完成标准

难度	题号与链接	核心考点	配套源码
[入门]	学校 OJ：1616 ｜ A 的 B 次方	整数快速幂、`1 % mod`、防溢出	`qpow` 模板
[基础]	学校 OJ：1617 ｜转圈游戏	轮数合并、$10^k \bmod n$ 快速幂	`code/pow_circle.cpp`
[基础]	学校 OJ：1642 ｜ Fibonacci 第 n 项	2×2 矩阵快速幂、$n\le 2$ 特判	`code/mat_pow_fib.cpp`
[提高]	学校 OJ：1647 ｜迷路	边权拆点、$9N\times 9N$ 邻接矩阵幂	拆点矩阵快速幂

完成标准

能脱稿写出带防溢出和 `1 % mod` 的快速幂；给定一个 k 阶线性递推式（含常数项），能独立构造转移矩阵并通过矩阵快速幂 AC 以上习题。

课后安排与下一章预告

- 复习：把三阶递推 $F_n = F_{n-1} + 2F_{n-2} + F_{n-3} + 5$ 手动写出 4×4 转移矩阵并手算验证 F_4。
- 练习：独立 AC 本章 4 道练习，重点体会《迷路》中“把 n 个点拆成 $9n$ 个点化边权为 1”的建模技巧。
- 下一章预告：第 2 章《素数、约数与筛法》——从 $\sqrt{n}$ 试除法到 $O(n)$ 欧拉线性筛，再到 $R \le 2^{31}$ 的超大区间筛法！

本章一句话

把指数按二进制拆成 $1, 2, 4, 8 \ldots$，无论是整数还是转移矩阵，都能在几十步内跨越十亿级甚至百亿亿级次数。

谢谢！