递推算法

先定义状态，再找前后关系

HGNU-ACM 实验室算法组

基础讲解 $+$ 3 道例题 $+$ 10 道练习题





本章目录 | 基础与第一个例题

什么是递推

状态、边界、转移、顺序

与递归、动态规划的关系

例 3.4：昆虫繁殖

本章目录 | 后两个例题与练习

例 3.5：位数问题

例 3.6：过河卒

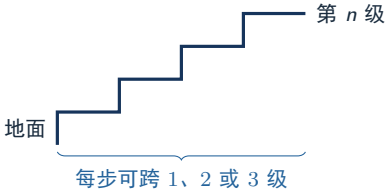
从例题走向常见题型

本章全部练习题索引

先看一个问题：上台阶

问题　楼梯有 n 级。每次可以上 1 级、2 级或 3 级，不能后退。问从地面走到第 n 级共有多少种走法。

- 走的顺序不同，就算不同方案。
- $n = 1$：只有 1 种；$n = 2$：$1 + 1$ 或 2，共 2 种。
- $n = 4$：可以先手算，不要急着写循环。



本页要点

关键不是“猜一个公式”，而是找：最后一步从哪里来。

把 n 很小时的答案都算出来

n	0	1	2	3	4	5	6
$f(n)$	1	1	2	4	7	13	24

其中 $f(0)=1$ 表示“原地不动”这一种空方案。

- $f(3)=4$：$111, 12, 21, 3$。
- $f(4)=7=4+2+1$。
- $f(5)=13=7+4+2$。
- $f(6)=24=13+7+4$。

观察

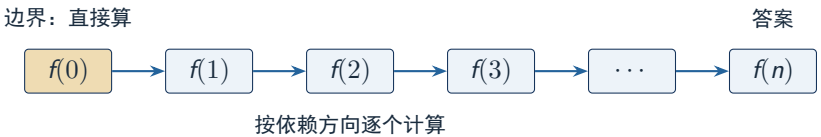
从 $n=3$ 开始，每一项都等于前面三项之和：

$$f(n)=f(n-1)+f(n-2)+f(n-3).$$

三个初始值 $f(0)=1, f(1)=1, f(2)=2$ 必须先写好。

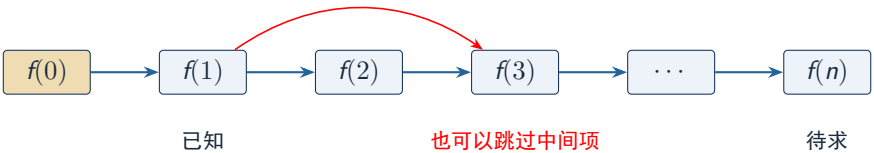
什么是递推

递推：把问题的答案表示成带下标的状态，再用已经求出的较小状态，按固定规则求出当前状态。



- 状态：$f(i)$ 表示什么，必须写清楚。
- 转移：$f(i)$ 由哪些更小的状态、按什么规则得到。
- 边界：最小的那些状态，不靠转移，直接给出答案。
- 顺序：保证计算 $f(i)$ 时，它依赖的状态已经算完。

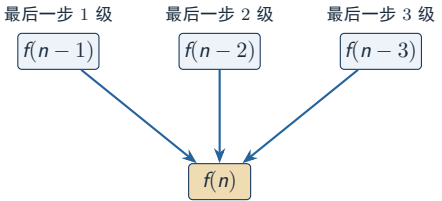
一维递推链：箭头就是计算方向



- 一维递推通常按 i 从小到大计算；箭头从已知指向未知。
- 每一步只依赖前面常数个状态时，时间通常是 $O(n)$。
- 先回答三个问题：从哪个下标开始？每一步依赖谁？算到哪个下标停？

转移从“最后一步”来

1. 把到达 $f(n)$ 的最后一步分成互不重叠的若干类。
2. 每一类都对应一个更小的状态。
3. 计数题把各类方案数相加；最优值题在候选中取最大或最小。



$$f(n) = f(n-1) + f(n-2) + f(n-3)$$

本页要点

“不重不漏”是转移式正确的第一道检查。

同一副骨架，换系数和边界

吃糖果

每天吃 1 块或 2 块，吃完 n 块的方案数：

$$f(n) = f(n-1) + f(n-2),$$

边界 $f(1) = 1, f(2) = 2$。

$$1,\ 2,\ 3,\ 5,\ 8, \ldots$$

Pell 数列

$$P(n) = 2P(n-1) + P(n-2),$$

边界 $P(1) = 1, P(2) = 2$。

$$1,\ 2,\ 5,\ 12,\ 29, \ldots$$

- 两个问题都属于“最后一步有两类”，所以都是两项相加。
- Pell 的系数 2 来自：前一个状态可以扩展出两种选择。
- 换题时先重做“状态、边界、转移”三件事，不要照抄代码。

递推算法的执行流程



1. 先用一句话写出状态的含义，再写转移式。
2. 用 $n = 1, 2, 3$ 的手算结果检查边界和转移。
3. 确认循环方向后，再写代码；代码只是把公式翻译一遍。
4. 最后用小样例、边界数据和最大值各测一次。

本页要点

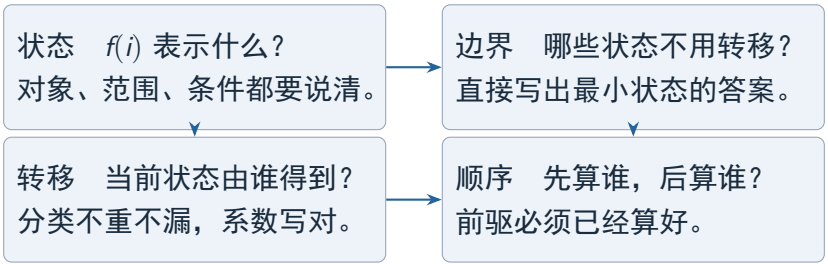
顺序：先想清楚，再写；不要边写循环边猜公式。

上台阶：核心代码

```
long long f[71] = {};
f[0] = 1;                       // 空方案：原地不动
f[1] = 1;
f[2] = 2;
for (int i = 3; i <= 70; ++i) {
    f[i] = f[i - 1] + f[i - 2] + f[i - 3];
}
```

- 循环从 $i = 3$ 开始，因为转移需要前三个状态。
- 每一项只依赖前 3 项，时间 $O(n)$，空间 $O(n)$。
- 本题 $n < 71$，答案在 `long long` 范围内；多组询问直接输出 $f[n]$。

递推四要素：缺一不可



- 四个部分互相对应：状态决定转移，边界决定起点，顺序保证转移可执行。
- 任何一个写错，后面的状态都会跟着错。

状态表示什么：一句话模板

模板：先用完整中文写一遍，再写公式

f[i] = 从 起点 到 位置/阶段，满足 条件 的 数量/最优值。

较好的定义

- $f(i)$：从地面走到第 i 级台阶的走法数。
- $dp(i,j)$：从 $(0,0)$ 走到 (i,j) 的合法路径数。

含糊的定义

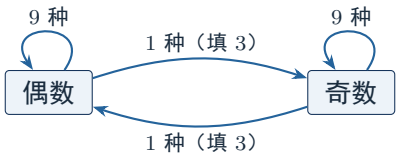
- $f(i)$：前 i 级的答案。
- $dp(i,j)$：到 (i,j) 的方案数。

- 没有写“起点”“障碍条件”，别人（以及未来的你）就无法判断转移是否完整。
- 同一个下标换一种含义，转移式就可能完全不同。

状态要保存“影响未来”的全部信息

例子：统计数字 3 出现偶数次的数。

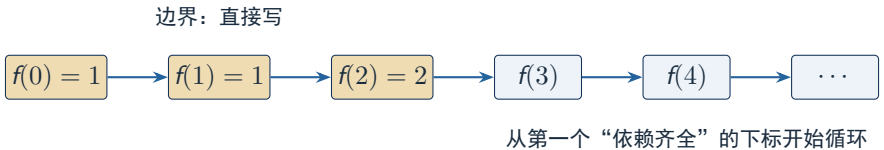
- 只记“偶数个 3”不够：再来一个 3 会变成奇数。
- 需要同时知道“奇数个 3”的数量。
- 状态取奇偶性，两个状态就够。



本页要点

状态不是越多越好，但不能少到丢失必要信息。

边界：不用递推就能得到的答案



- 计数题中，“什么都不做”通常算 1 种空方案，所以 $f(0) = 1$。
- 若某种状态不可能出现，可以设成 0；若求最小值且不可达，设成正无穷。
- 边界不能凭感觉：用题目的实际含义解释它，例如“站在地面”只有一种状态。

边界的常见类型与循环起点

类型	典型取值	例子
空方案	1	上台阶 $f(0)=1$
不可能	0	过河卒走到马控制点
不可达的最优值	$+\infty$	求最小代价但当前无路
固定前缀	按题意直接算	位数问题 $E_1=8, O_1=1$

- 循环必须从“所有前驱都已有定义”的第一个下标开始。
- 上台阶的转移依赖前三项，所以从 $i=3$ 开始循环；$f(0), f(1), f(2)$ 提前写好。
- 如果从 $i=1$ 开始却访问 $f(-2)$，程序不会自动帮你补 0，只会读到垃圾值。

自检

把循环第一轮代入转移式，逐个检查每个下标是否存在、是否已经赋值。

转移方程：把“最后一步”翻译成公式

按最后一步分类 → 每类对应一个前驱 → 计数相加 / 最值取优

计数型

上台阶：

$f(n) = f(n-1) + f(n-2) + f(n-3)$。

每个合法方案恰好属于“最后一步为 $1/2/3$ 级”中的一类。

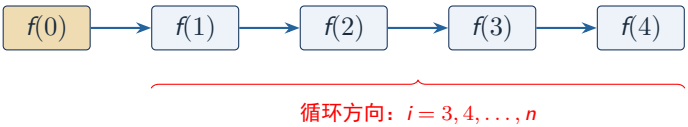
最优型

$$g(i) = \max\{g(i-1), g(i-2)\} + w_i.$$

每个候选代表一种决策，取最优值，不要把所有候选相加。

- 若一个前驱可以扩展出 k 种选择，转移中要乘上 k。
- 先检查“有没有漏类”，再检查“有没有重复类”。

计算顺序：前驱必须先算好



- 若 $f(i)$ 依赖 $f(i-1)$，就从小到大算；依赖方向反过来，就从大到小算。
- 一维顺序错了，通常会读到尚未赋值的元素；二维顺序错了，路径数会少算。
- 写循环前，把“当前状态依赖谁”列出来，顺序自然就确定了。

本页要点

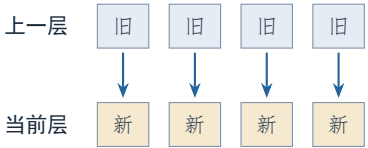
顺序不是习惯问题，而是正确性问题。

二维网格依赖：上、左两个前驱



- $dp[i][j]$ 依赖 $dp[i-1][j]$（上）和 $dp[i][j-1]$（左）。
- 按行优先填表时，两个前驱都已经算完。
- 第一行、第一列缺少一个前驱，通常单独初始化。
- 若循环方向相反，读到的 0 并不代表“没有路径”，而是“还没算”。

滚动数组：能省则省，但不能省错

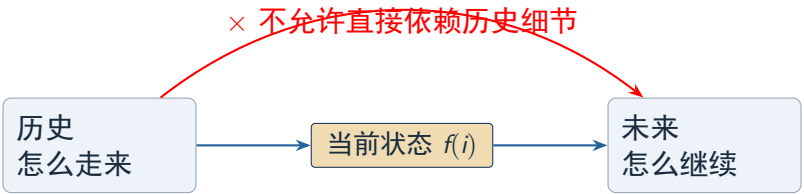


- 若当前层只依赖上一层，保留两行即可，空间从 $O(nm)$ 降到 $O(m)$。
- 若当前层还依赖左边（同一层），要保证左边先算，或者使用临时数组同步更新。
- 昆虫繁殖会用到 $i-x-2$，依赖距离不是常数，不能只留两行；需要保留足够长的历史。

本页要点

空间优化前，先画出完整的依赖箭头。

无后效性：未来只看现在的状态



- 无后效性：一旦当前状态确定，未来只由这个状态决定，与到达它的路径无关。
- 上台阶只关心“现在在第几级”，不需要记住之前每一步怎么走。
- 过河卒只关心“到达该格的路径数”，不需要保存每一条具体路径。
- 如果未来需要“上一步方向”等信息，就要把该信息并入状态。

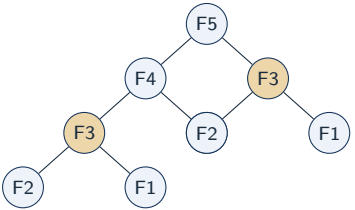
什么时候可以用递推



- 可定义：答案能用一个或几个有限的、可枚举的状态表示。
- 无后效性：状态值包含了决定未来的全部信息。
- 依赖无环：顺着“依赖谁”的箭头走，不会绕回自己；因此能排出先后。
- 可顺序计算：存在一个顺序，使每个状态被算时前驱都已完成。
- 子问题重叠是递推“划算”的原因；若子问题完全不重叠，分治通常更自然。

递归：从大问题追问小问题

- 递归函数直接按定义调用自身。
- 从 $F(5)$ 向下问，直到碰到边界 $F(1), F(2)$。
- 思路直观，但同一个子问题可能被重复计算。

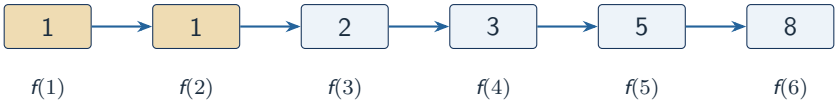


黄色节点是重复子问题

递归式

$$F(n) = F(n-1) + F(n-2),$$
$$F(1) = F(2) = 1.$$

递推：从小到大填表



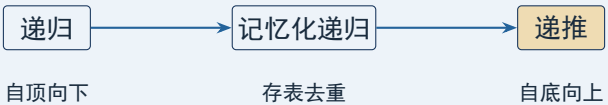
$$f(1) = 1, \quad f(2) = 1, \quad f(i) = f(i-1) + f(i-2) \quad (i \geq 3).$$

- 从边界开始，每个状态只算一次；没有函数调用栈，适合大规模数据。
- 同一组状态和转移：递归是“自顶向下”，递推是“自底向上”。
- 递归加记忆化，把算过的结果存起来，效果通常与递推相当。

递推、递归、动态规划不是三件互斥的事

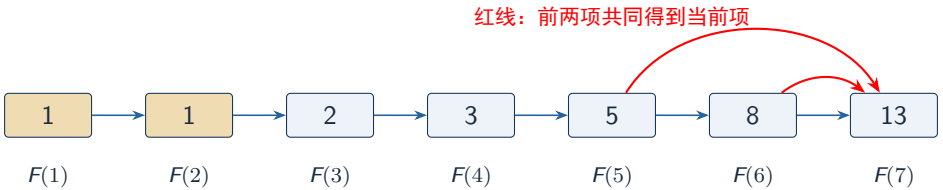
动态规划：状态设计 + 转移 + 边界 + 计算顺序

同一个转移方程的三种实现方式



- 动态规划是设计思想：怎样定义状态、怎样转移、按什么顺序算。
- 递推是常见实现：用循环把表从小下标填到大下标。
- 本章题目是动态规划的入门形态，后续会加入更复杂的状态和维度。

斐波那契：一维递推的标准图



$$F(1) = F(2) = 1, \qquad F(i) = F(i-1) + F(i-2) \quad (i \geq 3).$$

- 每个状态只依赖前两个状态，所以可以只保存两个变量，空间 $O(1)$。
- 若题目只要第 n 项且 n 很大，先看是否取模、是否要预处理多组询问。

斐波那契：预处理函数

```
const int MOD = 1000;
const int MAXA = 1000000;
int f[MAXA + 5] = {};                // 全局数组，避免栈溢出

void prepare() {
    f[1] = 1;                        // 边界
    f[2] = 1;
    for (int i = 3; i <= MAXA; ++i) {
        f[i] = (f[i - 1] + f[i - 2]) % MOD;  // 边算边取模
    }
}
```

- 预处理一次 $O(\max a)$，之后每次询问直接查表。

斐波那契：多组询问

```cpp
#include <iostream>
using namespace std;

int main() {
    prepare();
    int T;
    cin >> T;
    while (T--) {
        int a;
        cin >> a;
        cout << f[a] << '\n';       // 直接查表
    }
    return 0;
}
```

- 询问次数很多时，预处理比每组重新递推更省时间。
- 注意 $a = 1$ 的边界，输出 $f[1] = 1$。

复杂度：先数状态，再数转移

问题	状态数	每个状态的转移	时间 / 空间
上台阶	$O(n)$	$O(1)$	$O(n)$ / $O(n)$ 或 $O(1)$
斐波那契	$O(n)$	$O(1)$	$O(n)$ / $O(1)$
位数问题	$O(n)$（2 个状态）	$O(1)$	$O(n)$ / $O(1)$
昆虫繁殖	$O(z)$	$O(1)$	$O(z)$ / $O(z)$
过河卒	$O(nm)$	$O(1)$	$O(nm)$ / $O(nm)$

估算公式

$$\text{时间} \approx \text{状态数} \times \text{每个状态枚举的转移数}.$$

- 状态数不是“循环层数”的同义词；一维状态也可能有多个分量。
- 空间可以先按完整表估算，再判断能否滚动压缩。

取模与溢出：先算清上界，再选类型

每步都保持在范围内



不能等最后才取模

- `int` 约 2.1×10^9，`long long` 约 9.2×10^{18}。
- 加法后立刻取模；乘法先用 `long long` 承接。
- 负数取模：`(x % MOD + MOD) % MOD`。

$$f(i) = (f(i-1) + f(i-2)) \bmod M,$$
$$g(i) = ((\texttt{long long})a * b) \bmod M.$$

不够怎么办

若答案本身可能超过 `long long`，改用高精度整数（如 `cpp_int`），不能靠溢出“碰运气”。

四类常见错误：先定位，再修改

1. 边界漏
循环第一轮就读到未定义状态。

2. 顺序反
前驱还没算，当前状态已经填表。

3. 取模 / 溢出
中间值先超出类型范围，或负数余数错。

4. 状态不全
漏掉一类信息，导致重复或漏算。

- 先看第一个错误出现在哪个下标，再回溯它依赖的状态。
- 把出错的状态、边界、循环顺序写出来，比盯着代码猜更快。

边界漏与顺序反：两个最小例子

边界漏

上台阶若只写 $f(1)=1, f(2)=2$，循环从 $i=3$ 开始：

$$f(3)=f(2)+f(1)+f(0),$$

但 $f(0)$ 未定义。手算应是 4，程序可能得到任意值。

修正

补上 $f(0)=1$，并从 $i=3$ 开始循环。

顺序反

两位数奇偶状态若先改 even：

$$\texttt{odd} \leftarrow \texttt{even}_{\text{新}} + 9\texttt{odd},$$

就用了本轮的新值。

修正

先算两个 next，再统一赋值；或按依赖顺序更新。

取模 / 溢出与状态不全

- 先溢出后取模没有意义：溢出已经发生，取模救不回来。
- 模数为 12345 时，9×12344 仍在 `int` 范围内；但模数接近 10^9 时，乘法必须用 `long long`。
- 状态不全常表现为“答案偏大”或“某些情况永远算不到”。例如只记偶数个 3，就无法处理再来一个 3 的情况。
- 修复方法：把缺失的信息加入下标，或拆成多个互斥状态。

检查顺序

先确认状态是否完整，再检查边界，最后检查循环方向和数值类型。

写递推题的五项自查

1. 状态：每个下标、每个维度各表示什么？起点和条件写清了吗？
2. 边界：所有最小状态都有值吗？空方案、不可能、不可达分别取什么？
3. 转移：最后一步的分类是否不重不漏？系数是否正确？
4. 顺序：循环第一轮的前驱是否都已算好？同步更新是否需要临时变量？
5. 数值：是否每步取模？乘法是否溢出？答案是否需要高精度？

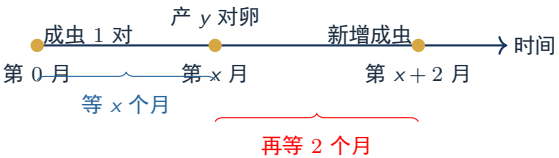
四行笔记

状态定义；初始值；转移式；循环顺序。写完这四行，再开始写代码。

例 3.4 | 昆虫繁殖：先画时间轴

题意

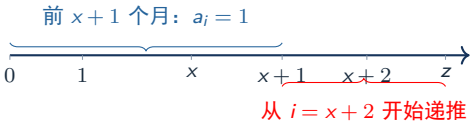
每对成虫过 x 个月产 y 对卵；每对卵再经过 2 个月长成成虫；成虫不死。第 0 个月有 1 对成虫，求第 z 个月的成虫对数。



- 新增成虫不能立即繁殖，要等 x 个月。
- 产卵和孵化各占一段时间，不能只算一次延迟。
- 约束：$0 \le x \le 20$，$1 \le y \le 20$，$x \le z \le 50$。

例 3.4 | 把“第几个月”编号清楚

- 为方便下标，把初始时刻记为第 0 个月。
- 第 0 个月到第 $x+1$ 个月，成虫数都还是 1：第一批新成虫尚未长成。
- 第一批新成虫在第 $x+2$ 个月出现。
- 从此每个月的新增成虫，来自两个月前产下的卵。



本页要点

下标从 0 开始时，转移的起点也要相应平移。

例 3.4 | 状态与新增成虫

- 定义 a_i：第 i 个月的成虫总对数。
- 第 i 个月新增的成虫，来自第 $i-2$ 个月产下的卵，数量为 $y \cdot a_{i-2}$ 吗？
- 不是：第 $i-2$ 个月能产卵的成虫，必须已经成熟至少 x 个月，数量可追溯到 a_{i-x-2}。

转移

$$a_i = a_{i-1} + y\,a_{i-x-2} \qquad (i \geq x+2).$$

- a_{i-1}：上个月已有的成虫，成虫不死。
- $y\,a_{i-x-2}$：本月新长成的成虫。
- 边界：$a_0 = a_1 = \cdots = a_{x+1} = 1$。

例 3.4 | 用样例手算一遍

取 $x = 1, y = 2$：

$$a_i = a_{i-1} + 2a_{i-3} \quad (i \geq 3).$$

i	0	1	2	3	4	5	6	7	8
a_i	1	1	1	3	5	7	13	23	37

样例输入 1 2 8，输出 37。

- $a_3 = a_2 + 2a_0 = 1 + 2 = 3$。
- $a_4 = a_3 + 2a_1 = 3 + 2 = 5$。
- $a_5 = a_4 + 2a_2 = 5 + 2 = 7$。
- 新增项分别出现在 $i = 3, 4, 5, \ldots$，与时间轴一致。

本页要点

边界与下标对齐，样例才能对上。

例 3.4 | 完整程序（1/2）：边界与类型

```cpp
#include <boost/multiprecision/cpp_int.hpp>
#include <iostream>
#include <vector>
using namespace std;
using boost::multiprecision::cpp_int;

int main() {
    int x, y, z;
    cin >> x >> y >> z;

    vector<cpp_int> a(z + 1, 1);    // 先全部置 1，代表边界
```

- a_i 可能远大于 `long long`，这里用高精度整数保存。
- `vector` 初始化为 1，正好对应前 $x+1$ 个月的边界。

例 3.4 | 完整程序（2/2）：转移与输出

```
    for (int i = x + 2; i <= z; ++i) {
        cpp_int newborn = y * a[i - x - 2];  // 本月新增
        a[i] = a[i - 1] + newborn;           // 旧成虫 + 新成虫
    }

    cout << a[z] << '\n';
    return 0;
}
```

- 循环从 $i = x + 2$ 开始，因为第一批新成虫此时才出现。
- 新增项的系数是 y：每对成虫产 y 对卵。
- `cpp_int` 的加法、乘法会自动扩展位数，不需要手动处理进位。

例 3.4 | 三个自测与易错点

1. $z < x + 2$：还没有新成虫，答案应为 1。
2. $y = 1$：检查是否把“卵数”误当成“新增成虫数”。
3. $x = 0$：最早的新成虫在第 2 个月出现，循环边界必须包含 2。

常见错误

- 把 a_{i-x-2} 写成 a_{i-x-1}，整体提前一个月。
- 用 `long long` 存答案；极端参数下会溢出。
- 忘记“卵长成后的第一个月不产卵”，导致重复计算。

本页要点

复杂度：时间 $O(z)$，空间 $O(z)$；本题重点是把时间轴翻译成下标。

例 3.5 | 位数问题：一个状态不够

题意

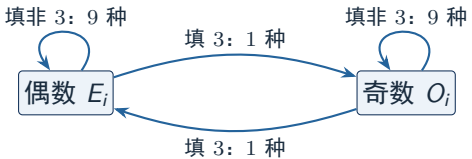
统计所有 N 位十进制数中，数字 3 出现偶数次的数的个数，答案对 12345 取模。$N \le 1000$。

- 只看“偶数个 3”不够：再添一位 3，偶数会变成奇数。
- 需要同时记录“奇数个 3”的数量，才能互相转移。
- 设 E_i：前 i 位中数字 3 出现偶数次的方案数；O_i：出现奇数次的方案数。

本页要点

状态必须覆盖所有会影响后续的情况；这里只差一个奇偶性。

例 3.5 | 奇偶状态机



$$E_i = 9E_{i-1} + O_{i-1},$$
$$O_i = E_{i-1} + 9O_{i-1}.$$

- 新位不是 3：奇偶性不变，有 9 种选法（包括 0）。
- 新位是 3：奇偶性翻转，只有 1 种选法。

例 3.5 | 首位为什么特殊

- N 位数的首位不能是 0。
- 首位选 3：1 种，进入奇数状态，所以 $O_1 = 1$。
- 首位选 $1, 2, 4, 5, 6, 7, 8, 9$：8 种，进入偶数状态，所以 $E_1 = 8$。
- 从第 2 位起可以填 0，非 3 的数字有 9 种。

本页要点

边界不是“$E_1 = 9$”：首位少了一个 0。

选 3：奇数



首位不能选 0

其余 8 个：偶数

例 3.5 | 样例 $N = 2$ 与边界检查

$$E_2 = 9E_1 + O_1 = 9 \times 8 + 1 = 73,$$
$$O_2 = E_1 + 9O_1 = 8 + 9 = 17.$$

- $E_2 + O_2 = 90$，正好是所有两位数 $10 \sim 99$ 的个数。
- 样例答案 73，与 E_2 一致。
- 若把首位也按 9 种非 3 处理，会多算以 0 开头的串。
- 若只更新 E 而忘记 O，下一位就无法继续转移。

复杂度

N 轮，每轮常数次运算：时间 $O(N)$，空间 $O(1)$。

例 3.5 | 完整程序（1/2）：边界

```cpp
#include <iostream>
using namespace std;

int main() {
    int n;
    cin >> n;

    const int MOD = 12345;
    int even = 8;                   // 首位非 3 的 8 种
    int odd = 1;                    // 首位是 3 的 1 种
```

- even 和 odd 分别保存当前长度的两个状态。
- 边界只算首位，所以非 3 的数字只有 8 个。

例 3.5 | 完整程序（2/2）：同步转移

```
    for (int i = 2; i <= n; ++i) {
        int next_even = (9 * even + odd) % MOD;
        int next_odd = (even + 9 * odd) % MOD;
        even = next_even;           // 先算完两个新状态
        odd = next_odd;             // 再统一赋值
    }
    cout << even % MOD << '\n';
    return 0;
}
```

- 若先写 `even = ...`，下一行的 `odd` 就会用到本轮的新值。
- 每次加法后立刻取模，两个状态都不会超出 `int`。
- 时间复杂度 $O(N)$，空间 $O(1)$。

例 3.5 | 状态压缩与易错点

- 第 i 层只依赖第 $i-1$ 层，所以不需要保存所有 N 层。
- 两个旧状态、两个新状态，共四个整数即可完成一轮更新。
- 如果统计“出现次数模 k 为某值”的类别，就要 k 个状态，而不是两个。

易错点

- 首位按 9 种非 3 处理（多算了以 0 开头的串）。
- 忘记奇数状态，导致转移不闭合。
- 先改 `even` 再算 `odd`（顺序反了）。
- 只在最后取模，中间值增长过大。

例 3.6 | 过河卒：把棋盘变成状态表

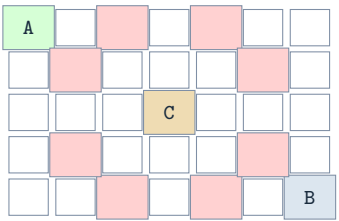
题意

卒从 $A(0,0)$ 走到 $B(n,m)$，每步只能向右或向下。马所在的格及其一步可达的八个格不能经过。求合法路径数。

$n, m \le 20$。

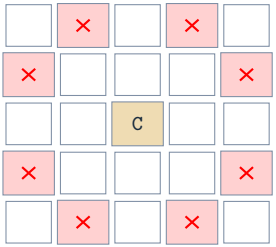
- 一个格子最多从上方或左方进入。
- 被马控制的格子路径数固定为 0。
- 路径只能向右、向下，因此依赖关系没有环。

C 为马的位置，红格为控制点



例 3.6 | 马的控制点：八个偏移

- 马走“日”字，偏移为 $(\pm 1, \pm 2)$ 与 $(\pm 2, \pm 1)$，共 8 个。
- 先标记马所在格，再枚举八个偏移。
- 只标记仍在棋盘内的格子；越界坐标直接跳过。
- 起点和终点保证不是控制点，但仍建议用统一的边界处理。



马在中心时的八个控制点（示意图）

本页要点

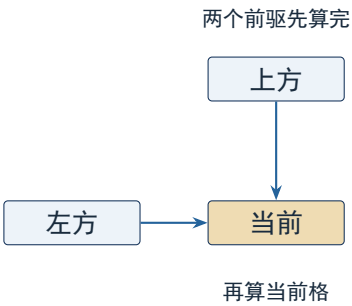
漏掉任意一个控制点，都会把障碍格当成可走格。

例 3.6 | 状态与依赖

- 定义 $dp[i][j]$：从 $(0,0)$ 走到 (i,j) 的合法路径数。
- 若 (i,j) 被控制，$dp[i][j]=0$。
- 否则，最后一步只能从上方或左方来：

$$dp[i][j]=dp[i-1][j]+dp[i][j-1].$$

- 越界的前驱按 0 处理。



例 3.6 | 边界、顺序与样例

- 起点：若 $(0,0)$ 未被控制，$dp[0][0]=1$，表示“还没走”的空路径。
- 第一行只能从左来，第一列只能从上来；可以显式初始化。
- 其余格按 $i=1\ldots n$、$j=1\ldots m$ 的顺序计算。
- 样例 $n=8, m=6, C=(0,4)$，答案为 1617。
- 无阻挡时 $dp[n][m]=\binom{n+m}{n}$，可以用小棋盘独立核对。

数值范围

最大路径数可能超过 `int`，使用 `long long`；本题棋盘只有 21×21，不会超过 `long long`。

例 3.6 | 程序（1/3）：输入与障碍

```cpp
#include <iostream>
using namespace std;

int main() {
    int n, m, cx, cy;
    cin >> n >> m >> cx >> cy;

    bool blocked[25][25] = {};
    long long dp[25][25] = {};
    int dx[8] = {1, 1, -1, -1, 2, 2, -2, -2};
    int dy[8] = {2, -2, 2, -2, 1, -1, 1, -1};

    blocked[cx][cy] = true;          // 马所在格
```

- 数组开 25×25，覆盖 $n, m \le 20$ 并留出安全余量。
- `dx`,`dy` 一一配对，表示马的八个跳跃方向。

例 3.6 | 程序（2/3）：标记控制点与边界

```
for (int k = 0; k < 8; ++k) {
    int x = cx + dx[k];
    int y = cy + dy[k];
    if (0 <= x && x <= n && 0 <= y && y <= m) {
        blocked[x][y] = true;  // 棋盘内的控制点
    }
}

dp[0][0] = blocked[0][0] ? 0 : 1;  // 起点空路径
for (int i = 1; i <= n; ++i) {
    dp[i][0] = blocked[i][0] ? 0 : dp[i - 1][0];
}
for (int j = 1; j <= m; ++j) {
    dp[0][j] = blocked[0][j] ? 0 : dp[0][j - 1];
}
```

- 越界的控制点不标记；边界行、列只有一个前驱，单独处理。

例 3.6 | 程序（3/3）：填表与输出

```
    for (int i = 1; i <= n; ++i) {
        for (int j = 1; j <= m; ++j) {
            if (blocked[i][j]) {
                dp[i][j] = 0;       // 马控制点不可达
            } else {
                dp[i][j] = dp[i - 1][j] + dp[i][j - 1];
            }
        }
    }

    cout << dp[n][m] << '\n';
    return 0;
}
```

- 每个内部格的两个前驱都已算好，顺序正确。
- 若终点被控制，答案为 0；本题保证 $C \neq B$，但代码仍能处理。

例 3.6 | 核对、复杂度与易错点

- 样例：$8\ 6\ 0\ 4 \Rightarrow 1617$。
- 时间 $O(nm)$，空间 $O(nm)$；最多 21×21 个状态。
- 可用无阻挡棋盘检验：$dp[n][m] = \binom{n+m}{n}$。

易错点

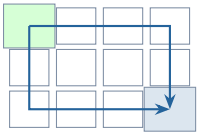
- 只标记马所在格，漏掉八个控制点。
- 第一行、第一列没有单独初始化，或把越界前驱当成 1。
- 用 `int` 存路径数，在较大棋盘上溢出。
- 循环方向反了，读到尚未计算的 dp。

一维题型：上台阶、斐波那契、Pell

题目	边界	转移	前几项
上台阶	$f(0)=1, f(1)=1, f(2)=2$	$f(n)=\sum_{k=1}^{3} f(n-k)$	$1,1,2,4,7,\ldots$
斐波那契	$F(1)=F(2)=1$	$F(n)=F(n-1)+F(n-2)$	$1,1,2,3,5,\ldots$
Pell	$P(1)=1, P(2)=2$	$P(n)=2P(n-1)+P(n-2)$	$1,2,5,12,29,\ldots$
吃糖果	$f(1)=1, f(2)=2$	$f(n)=f(n-1)+f(n-2)$	$1,2,3,5,8,\ldots$

- 做法完全同构：定状态、写边界、按最后一步分类。
- 不同点只在系数、边界和是否取模；不要背题，要背“四要素”的检查流程。
- 多组询问时，先预处理到最大下标，再 $O(1)$ 回答。

二维题型：移动路线与流感传染



移动路线：右 / 下，格点计数

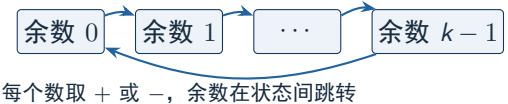
- 移动路线：
  $dp[i][j] = dp[i-1][j] + dp[i][j-1]$，
  与过河卒同型。
- 流感传染：第 d 天的新感染来自第 $d-1$ 天，必须按天同步更新。
- 同一天内不能让新感染继续传染，否则会“当天连锁”。

判断依赖

上一层用两层数组；同一层还依赖左边/上边时，按正确顺序填二维表。

多状态与组合计数：放苹果、判断整除

- 放苹果：把 m 个相同苹果放入 n 个相同盘子，允许空盘。按“是否有空盘”分类，练边界与二维状态。
- 判断整除：处理前 i 个数时，记录可达的模 k 余数；每个数可取 $+$ 或 $-$，是“多状态滚动”的典型。
- 两题都要先写清楚状态中的“前 i 个”与“当前余数/盘数”各表示什么。



挑战题：山区建小学

- 先预处理 $cost(l, r)$：第 l 到第 r 个村庄只建一所学校的最小距离和。
- 再设 $g[i][j]$：前 i 个村庄建 j 所学校的最小总距离。
- 转移枚举最后一所学校负责的区间：

$$g[i][j] = \min_{j-1 \le k < i} \{g[k][j-1] + cost(k+1, i)\}.$$

- 这是“区间预处理 + 二维递推”的挑战题，先完成前面的基础题再来做。

练习顺序

一维数量题 $\rightarrow$ 二维路径/同步更新题 $\rightarrow$ 多状态与区间代价题。

练习安排：由一维到二维

1. 先做菲波那契、Pell、上台阶，熟悉边界与取模。
2. 再做流感传染、移动路线，训练二维状态与同步更新。
3. 最后做放苹果、吃糖果、判断整除、踩方格和山区建小学。

每题的笔记格式

写四行：状态定义、初始值、转移式、循环顺序；代码写完再用样例和小规模手算核对。

本章练习题索引 | 1–5 / 10

1. P5185 | 菲波那契数列 (2)（一本通 1188）
2. P5186 | Pell 数列（一本通 1189）
3. P5187 | 上台阶（一本通 1190）
4. P5188 | 流感传染（一本通 1191）
5. P5189 | 放苹果（一本通 1192）

点击蓝色题名进入学校 OJ；本章截图中的练习全部列在索引中。

本章练习题索引 | 6–10 / 10

6. P5190 | 吃糖果（一本通 1193）
7. P5191 | 移动路线（一本通 1194）
8. P5192 | 判断整除（一本通 1195）
9. P5193 | 踩方格（一本通 1196）
10. P5194 | 山区建小学（一本通 1197）

点击蓝色题名进入学校 OJ；本章截图中的练习全部列在索引中。

收尾：交卷前检查这五件事

1. 状态定义能否用一句完整的中文说清？
2. 边界的每个值能否解释实际含义？
3. 转移是否覆盖所有情况、没有重复？
4. 循环第一轮的前驱是否全部已赋值？
5. 取模、溢出、多组数据是否都考虑？

下一章

递归算法会把“从大问题追问小问题”展开；带着本章的状态与边界意识去学，会更容易理解记忆化搜索。