状压与数位 DP

用二进制整数压缩整行状态 + 按数位记忆化统计区间整数

HGNU-ACM 实验室算法组

承接第 3 章《树形与换根 DP》丨配套位运算工具箱、数位框架与 3 道真题精讲

本章目录

状态压缩 DP：位运算工具箱与棋盘压行

数位 DP：记忆化搜索框架

前两节零基础拆解：先学“位运算工具箱与棋盘压行”，再学“数位 DP 记忆化框架 `dfs(pos, pre, lead, limit)`”。

本章目录（续）

状压 DP 真题精讲

数位 DP 真题精讲

常见错误、检查清单与练习安排

后三节精讲《炮兵阵地》《国王》《Windy 数》三道真题，并给出常见错误对照与提交前检查清单。

生活引入：从“树上选点”到“棋盘排兵”

承接第 3 章

上一章《树形与换根 DP》中，每个节点只有“选”或“不选”两种状态，我们用 `dp[u][0/1]` 记录一维选择。

- 现在把场景搬到 $N \times M$ 的棋盘上：第 i 行有 M 个格子（例如 $M = 10$），每格要决定放（1）或不放（0）士兵。
- 下一行能不能放，取决于上一行每一列有没有放——如果用数组传状态，得写 `dp[row][c0][c1]...[c9]`，足足 11 维！
- 换个思路：一排 10 个开关，开记为 1，关记为 0，合起来不就是一个 10 位二进制数吗？

本页要点

当一行只有 $M \le 12$ 个“选/不选”开关时，整行状态可以直接压成一个普通的 `int` 整数！

暴力为什么会 TLE？

- 假设棋盘有 $N = 10$ 行、$M = 10$ 列，总共 100 个格子，问有多少种不冲突的摆放方案。
- 暴力 **DFS** 逐格搜索：每个格子枚举“放/不放”，最多分支 $2^{100} \approx 1.26 \times 10^{30}$ 次，宇宙毁灭也算不完，必 **TLE**。
- 按行压成整数：每行只有 $2^{10} = 1024$ 种状态；先筛掉左右相邻冲突的行，合法状态仅剩 $k \le 144$ 个（甚至 60 个）。
- 逐行转移总次数仅约 $N \times k^2 \approx 10 \times 144^2 \approx 2 \times 10^5$ 次，耗时不到 1 毫秒！

本页要点

暴力按格子指数爆炸；状压把“一格格猜”变成“一行行整块转移”，复杂度瞬间降到可接受范围。

新术语大白话：状态压缩与位掩码

两个核心术语

- 状态压缩（**Bitmask DP**）：把本来需要用一排布尔数组 `bool a[10]` 表示的集合或行状态，按二进制位拼成一个整数 `s`。
- 位掩码（**bitmask**）：这个整数 `s` 就叫位掩码。它的第 j 位（从右往左数，最低位是第 0 位）是 1 表示“第 j 列选了”，是 0 表示“没选”。

- 例如 $M = 4$，二进制 `1010` 表示第 1 列和第 3 列放了士兵，第 0 列和第 2 列空着。
- 十进制对应 $1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 10$，所以整数 `s = 10` 就代表这一整行！

本页要点

位掩码的本质就是“十进制整数的二进制外壳”，一个 `int` 就能装下最多 31 个开关。

位运算工具箱（一）：单点查询与修改

需求	位运算写法	手算例子（设 s = 10，即 1010）
取第 i 位是不是 1	`(s >> i) & 1`	$i=1$：(1010 >> 1) & 1 = 1
把第 i 位置为 1	`s | (1 << i)`	$i=0$：1010 | 0001 = 1011（11）
把第 i 位清成 0	`s & ~(1 << i)`	$i=1$：1010 & 1101 = 1000（8）
把第 i 位 0/1 翻转	`s ^ (1 << i)`	$i=2$：1010 ^ 0100 = 1110（14）

- 记住右移 `>> i` 相当于把第 i 位搬到最右边第 0 位，再 `& 1` 就只看这一位。

本页要点

单点操作全靠 `1 << i` 造出“只有第 i 位是 1 的探针”，再配合 `&`、`|`、`^`。

位运算工具箱（二）：判冲突、数 1 与枚举子集

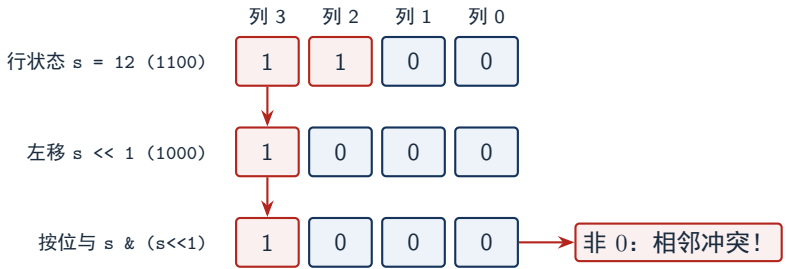
需求	位运算写法	为什么能这样写？
两行 a, b 有无同列冲突	`(a & b) != 0`	只要某列同为 1，按位与结果就非 0
同一行左右相邻是否冲突	`(s & (s << 1)) != 0`	整体左移 1 位若和原数撞上，说明有相邻的 1
统计 s 中有多少个 1	`__builtin_popcount(s)`	GCC 内置指令，$O(1)$ 数出放了多少个
只枚举 s 的所有非空子集	`for(int sub=s; sub; sub=(sub-1)&s)`	每次减 1 借位再 `&` s，跳过不在 s 里的位

- 判左移 2 格冲突同理：`(s & (s << 2)) == 0`。枚举子集总复杂度是 $O(3^M)$，远小于 $O(4^M)$！

本页要点

用整体位运算替代 `for` 循环逐列检查，既快又不容易写错下标。

图解：棋盘一行的二进制编码与相邻冲突检查



本页要点

把整行右移/左移一位再做按位与，只要结果非 0 就说明至少有一对相邻格子同时为 1。

预处理合法行掩码 ok[]：砍掉九成无效转移

- 棋盘宽度 $M = 10$ 时，$0 \sim 2^{10} - 1$ 一共有 1024 个整数。
- 但绝大多数整数自身就已经违反了行内规则（比如存在相邻的 1，或者左右两格都有炮兵）。
- 如果在转移循环里硬枚举 1024×1024 对状态，不仅浪费时间，还容易把非法状态当合法转移。
- 标准做法：在 DP 开始前，先写一个循环把所有满足行内约束的整数存进 `ok[]`（或 `masks[]`）数组，同时记录它的 1 的个数 `cnt[]`。
- 例如炮兵阵地 $M = 10$ 时，1024 个状态预处理后只剩 60 个，后续所有转移只遍历这 60 个！

本页要点

先预处理 `ok[]` 筛掉行内非法状态，是棋盘状压 DP 提速与防错的第一步。

手算跟踪表：$M = 3$ 棋盘预处理与两行转移

小数据设定

设 $N = 2, M = 3$，同行及上下同列不能相邻，求第 2 行放满后的总方案数。

步骤	动作 / 枚举状态	得到结果	备注
1	枚举 $s = 0 \ldots 7$ 筛行内不相邻	`ok = {0, 1, 2, 4, 5}`	筛掉 3，6，7
2	初始化第 0 行前虚拟空行	`dp[0] = 1`，其余 0	从空态 0(000) 起步
3	第 1 行可取 `ok` 中任意 s	`dp[s] = 1`（共 5 态）	上一行是 0 全不冲突
4	第 2 行取 $s = 5$（101）	兼容上行 0，2	`dp'[5] = 2`
5	第 2 行取其余 $s = 0, 1, 2, 4$	累加各自兼容的上行	共 $5 + 4 + 4 + 4 + 2 = 19$
6（终止）	`row = 2 == N` 结束循环	累加末行全部 `dp`	输出最终方案数 19

本页要点

每步只看“当前行”与“上一行”是否冲突，手算也能核对到最后一行。

正确性论证：按行转移的不变量

阶段	状态不变量：处理完前 r 行后，dp[i] 等于末行为 ok[i] 的全部合法方案数
初始化	设虚拟第 0 行状态为 0（全空），dp[0] = 1，其余为 0，对应 0 行时唯一空棋盘。
保持	若第 r 行选 ok[i]，第 $r-1$ 行选 ok[j] 且行间不冲突，累加 dp[j] 即覆盖所有前 r 行合法组合，无后效性。
终止	当 $r = N$ 退出循环时，把最后一行所有合法掩码的 dp[i] 相加（或取最大值），即全棋盘答案。

本页要点

棋盘行与行之间只依赖相邻一两行，把整行压成整数后，按行推进天然满足无后效性。

模板代码（1/2）：预处理合法行掩码与初始化

```
int n, m; cin >> n >> m;                    // n 行 m 列
vector<int> ok, cnt;
for (int s = 0; s < (1 << m); ++s) {        // 枚举该行全部 2^m 种状态
    if ((s & (s << 1)) == 0) {              // 必须加外括号：检查左右是否相邻
        ok.push_back(s);                    // 存入合法行掩码
        cnt.push_back(__builtin_popcount(s)); // 记录这一行放了几个
    }
}
int k = (int)ok.size();                     // 合法状态总数远小于 2^m
vector<long long> dp(k, 0);
dp[0] = 1;                                  // ok[0] 必为 0，作为第 1 行的空跳板
```

本页要点

注意 `ok[0]` 一定是 0（全不放），因为循环从 `s = 0` 开始且 0 一定合法。

模板代码（2/2）：逐行滚动转移主循环

```
for (int row = 0; row < n; ++row) {          // 从第 0 行逐行往下推
    vector<long long> nxt(k, 0);             // 新一行状态数组（滚动防串行）
    for (int i = 0; i < k; ++i) {            // 枚举当前行选第 i 个合法掩码
        for (int j = 0; j < k; ++j) {        // 枚举上一行选第 j 个合法掩码
            if ((ok[i] & ok[j]) == 0)        // 上下同列不能同时为 1
                nxt[i] += dp[j];             // 合法则累加方案数
        }
    }
    dp.swap(nxt);                            // 滚动覆盖，省空间又防脏数据
}
long long ans = 0; for (long long v : dp) ans += v; // 汇总末行方案数
```

本页要点

用 nxt 算完一整行再 dp.swap(nxt)，省内存且不会把本行混进上一行。

深入细节：为什么（sub - 1）& s 只遍历子集？

手算例子：设 s = 1010（第 1、3 位为 1），只枚举它的非空子集

轮次	当前 sub	sub - 1	（sub - 1）& s（下一子集）
1	1010	1001	1001 & 1010 = 1000
2	1000	0111	0111 & 1010 = 0010
3	0010	0001	0001 & 1010 = 0000（终止）

- 每次减 1 会把最右边的 1 变成 0、更低位全变成 1；再 & s 就把不在 s 里的位瞬间抹掉！

本页要点

如果对所有 $s \in [0, 2^M - 1]$ 都用此法枚举子集，总次数恰好是 $\sum \binom{M}{k} 2^k = 3^M$，比 4^M 快得多！

空间压缩：滚动数组 vs 完整二维表

- 若开完整表 `dp[100][60][60]`，虽然本题够用，但遇 $N = 10^4$ 或状态更多时容易 **MLE**（超内存）。
- 观察转移方程：计算第 `row` 行时，只读取上一行（或前两行），更早的历史行再也不会用到！
- 因此只需保留当前行 `nxt` 和上一行 `dp`，每次用 `dp.swap(nxt)` 交换指针（$O(1)$ 时间）。
- 既把空间从 $O(N \cdot k^2)$ 压成 $O(k^2)$，又避免覆盖掉还没读完的上一行旧值。

本页要点

`dp.swap(nxt)` 是状压 DP 最干净的滚动写法：不手写 `& 1` 奇偶下标，绝不写错行号。

棋盘状压的两种依赖结构对比

模型类型	状态定义	代表例题
单行依赖（只受上一行影响）	`dp[i]`（或 `dp[i][c]`），只需枚举本行 i 与上行 j	国王（八邻域只影响相邻 1 行）
双行依赖（受前两行影响）	`dp[i][j]`，枚举本行 i、上行 j、上上行 p	炮兵阵地（攻击上下各 2 格）
含地形禁止	在枚举 `i` 时先判 `(masks[i] & hill[row]) == 0`	炮兵阵地（山地不可放炮）

本页要点

做题先看“纵向影响几行”：影响 1 行记一维掩码，影响 2 行记两维掩码，有障碍物就按位与过滤。

上机验证与反事实追问

3 组标准验证（以“同行及上下同列不相邻放士兵”计数为例）

1. 最小边界：$N = 1, M = 1 \to$ 合法掩码 $\{0, 1\}$，预期输出 2（空或放 1 个）。
2. 小数据手算组：$N = 2, M = 3 \to$ 对应前面手算表，预期输出 19。
3. 特殊情况：$N = 1, M = 4 \to$ 斐波那契规律 $F_6 = 8$，预期输出 8。

反事实追问

如果把代码里的 `if ((s & (s << 1)) == 0)` 漏掉外括号，写成 `if (s & (s << 1) == 0)` 会发生什么？

$\Rightarrow$ `==` 优先级高于 `&`！编译器会先算 `(s << 1) == 0`，导致几乎所有掩码被判非法，结果严重偏小（**WA**）。

本页要点

写位运算条件时，养成“只要出现 `&` 或 `|`，外层必须套一对小括号”的习惯。

状压 DP 易错对照：WA / TLE / RE 定位

错误类型	常见错误写法	正确做法与原因
WA（优先级）	`if (a & b == 0)`	必须写 `(a & b) == 0`，`==` 比 `&` 先算
WA（不可达）	求最值时 `dp` 初值全设 0	凑不出的状态不应转移，最值类不可达必设 `-1` 或 $-\infty$
TLE（未预处理）	转移内层直接枚举 $0 \sim 2^M - 1$	先用 `ok[]` 筛掉行内非法状态，内层只跑 k 次
RE（左移溢出）	`1 << 31` 对有符号 `int` 左移	超过 30 位要用 `1LL << i`，但状压通常 $M \le 20$

本页要点

状压 WA 八成是位与忘加括号或不可达状态没设 -1；TLE 八成是没先预处理合法掩码。

生活引入：怎么数 $[1, 10^9]$ 里不含 4 的数字？

- 车牌号或楼层号常常避开数字 4。如果问你：从 1 到 10^9 之间，有多少个整数的十进制表示里完全没有数字 4？
- 从 1 到 10^9 一个个转成字符串检查？十亿次循环直接超时！
- 换个思路：一个 9 位数不过就是往 9 个格子里填 $0 \sim 9$（不能填 4），从最高位往最低位填。
- 而且求区间 $[L, R]$ 内的个数，只需用前缀和思想拆成两半：

$$\text{答案} = \text{从}0\text{ 到}R\text{ 的好数个数} - \text{从}0\text{ 到}L-1\text{ 的好数个数}$$

本页要点

数位 DP 把“遍历几十亿个整数”变成“逐位填 $0 \sim 9$ 的记忆化搜索”，复杂度只与位数（≤ 20）有关。

暴力为什么会 TLE？

- 以《Windy 数》为例，数据范围是 $1 \le A \le B \le 2 \times 10^9$。
- 暴力枚举：写一个 `for (int x = A; x <= B; ++x)`，拆出每位数字检查相邻差是否 ≥ 2。
- 最坏要做 $2 \times 10^9 \times 10 = 2 \times 10^{10}$ 次运算，远超 1 秒上限（约 10^8 次），必 **TLE**。
- 数位记忆化搜索：数字最多只有 10 位，每位只有 10 种选择，上一位数字最多 11 种状态。
- 总状态数仅约 $10 \times 11 = 110$ 个，每个状态算 10 次，总共约 1100 次运算！

本页要点

区间长度是 10^9 甚至 10^{18}，但它的“位数”只有 $10 \sim 19$ 位，这就是数位 DP 快到飞起的原因。

新术语大白话：limit 与 lead

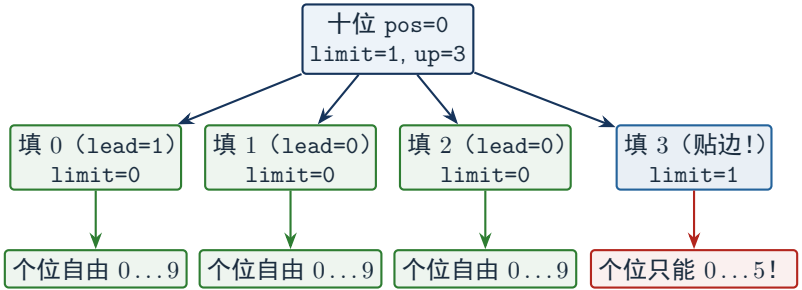
dfs(pos, pre, lead, limit) 的四个参数

- pos：当前正准备填第几位（从最高位 0 开始，到最低位 len - 1 结束）。
- pre：上一位填的数字是什么（用于判断相邻关系，如 $|d - \text{pre}| \geq 2$）。
- limit（上界贴合）：前面已经填的所有位，是不是全都卡死在上界上？若是，当前位最多只能填到 s[pos]-'0'；若前面某位填小了，后面就可以自由填 $0 \sim 9$！
- lead（前导零标记）：前面填的全是 0 吗？只要还在填前导零（如把 35 写成 00035），这些开头的 0 只是占位符，不算真正数字，不应受题目规则限制。

本页要点

limit 管“不能超过上界 X”，lead 管“开头的 0 不算正式数位”。

图解：数位贴边树（以求 $0 \sim 35$ 为例）



本页要点

只有一路选最大数字才会继续 `limit=1`；一旦某位选小了，整棵子树立刻进入 `limit=0` 自由态！

为什么必须有 limit 与 lead？（反例对比）

- 反例 **1**（没有 limit）：求 $0 \sim 35$ 的数时，十位填了 3，如果个位还按 $0 \sim 9$ 枚举，就会把 $36, 37, 38, 39$ 全部多算进去！
- 反例 **2**（没有 lead）：在《Windy 数》（要求相邻数位差 ≥ 2）中，考虑一位数 1。
  - 在两位数视角下它写成 01。
  - 如果不标记 lead=1，程序会以为前一位是真正的数字 0，检查 $|1-0| = 1 < 2$，直接把 1 给枪毙掉！
  - 实际上 1 只有一位，根本没有“前一位”，所以前导零阶段必须靠 lead 放行！

本页要点

没有 limit 会超界多算；没有 lead 会把一位数、两位数的开头 0 当成真实数字而误杀。

为什么只有 !limit && !lead 才能读写 memo？

limit	lead	能否读写 memo[pos][pre]？为什么？
0	0	可以！后面完全自由（$0 \ldots 9$），且前面已有真实数字，结果只由（pos，pre）决定。
1	0	不能！贴紧上界时，上界 35 和上界 39 在同一位置能填的上限不同，不能混用缓存。
0	1	不能！还在前导零阶段，pre 是哨兵或 0，与真实填过 0 的状态含义不同。
1	1	不能！既贴上界又在前导零，最多只会出现 1 次（整条最左/贴边路径），无需缓存。

本页要点

铁律：贴上界（limit=1）或还在前导零（lead=1）时，绝不读写 memo！

手算跟踪表：用 solve(23) 推演不含数字 4 的计数

小数据设定

统计 $[0, 23]$ 中不含数字 4 的整数个数（应为 $24 - 2 = 22$ 个，去掉 $4, 14$）。

步	调用 dfs(pos,lead,lim)	枚举 d	返回	说明
1	dfs(0, 1, 1)（up=2）	0	调 pos=1	十位填 0，仍前导零且脱离上界
2	dfs(1, 1, 0)（up=9）	$0 \ldots 9 \setminus \{4\}$	9	个位可填除 4 外的 9 个数字
3	回到 pos=0	1	9	调 dfs(1,0,0) 写 memo[1]=9
4	回到 pos=0	2	调 pos=1	十位填 2 贴边，limit 保持 1
5	dfs(1, 0, 1)（up=3）	$0, 1, 2, 3$	4	个位只能填 $0 \ldots 3$，无 4
6（止）	pos == 2 触底返回	—	22	总和 $9 + 9 + 4 = 22$，结束

本页要点

第 3 步脱离上界后写缓存，后面所有同类自由态就能直接 $O(1)$ 复用。

正确性论证：数位前缀等价性

阶段	递归不变量：`dfs(pos, pre, lead, limit)` 返回从 `pos` 到末位的所有合法填法数
初始化	从 `pos = 0`，`lead = 1`，`limit = 1` 出发，覆盖 $0 \sim X$ 的所有位数与上界约束。
保持	枚举当前位 $d \in [0, \mathrm{up}]$，跳过非法数字，将新的 `pre`，`lead && (d==0)`，`limit && (d==up)` 传入 `pos+1`，子问题彼此独立且不重不漏。
终止	当 `pos == len` 时说明所有数位填完，返回 1（若题目要求正整数则需特判 `!lead`）。

本页要点

每位按 $d = 0 \ldots \mathrm{up}$ 严格划分互不相交的子树，保证既不漏掉任何合法数，也不重复计算。

模板代码（1/2）：solve(x) 与 dfs 入口、边界

```
int a[22], len;
long long memo[22][12];                    // 记忆化数组，初始化为 -1
long long dfs(int pos, int pre, bool lead, bool limit) {
    if (pos == len) return !lead;          // 填完所有位：非全零则算 1 个合法正整数
    if (!limit && !lead && memo[pos][pre] != -1)
        return memo[pos][pre];             // 仅在不贴上界且非前导零时读缓存
    int up = limit ? a[pos] : 9;           // 贴上界最多到 a[pos]，否则到 9
    long long res = 0;
```

本页要点

递归边界如果允许统计 0 就直接 `return 1;`；如果只统计正整数（如 Windy 数），就返回 `!lead`。

模板代码（2/2）：枚举当前位、状态传递与写缓存

```
    for (int d = 0; d <= up; ++d) {
        if (!lead && 条件不满足) continue; // 前导零期间不触发限制
        int nxt = (lead && d == 0) ? 0 : d;
        res += dfs(pos + 1, nxt, lead && (d == 0), limit && (d == up));
    }
    if (!limit && !lead) memo[pos][pre] = res; // 只缓存自由且非前导零
    return res;
}
long long solve(long long x) {
    if (x <= 0) return 0;                    // 防 L=0 时 L-1 变负
    string s = to_string(x); len = s.size();
    for (int i = 0; i < len; ++i) a[i] = s[i] - '0';
    memset(memo, -1, sizeof(memo)); return dfs(0, 0, true, true);
}
```

写法对比：独立 bool lead vs 哨兵 last = 10

对比项	独立参数 bool lead	哨兵写法 last = 10
参数个数	多一个 bool lead（4 个参数）	把前导零塞进 last=10（3 个参数）
判断当前在填前导零	直接看 lead	看 last == 10
写缓存条件	if (!limit && !lead)	if (!limit)（last=10 仅 1 条路径）
适用场景	通用最强，哪怕数字和、统计出现次数也能用	相邻数位差、不能出现 62 等仅看前一位的题

本页要点

两种写法完全等价：初学者建议先练熟通用的 dfs(pos, pre, lead, limit)，看懂后再用哨兵精简。

数位 DP 变形：如何记录“数字和”或“模 K 余数”？

- 并不是所有数位 DP 都只关心“上一位数字 pre”。遇到下列题型，只需把 pre 换成对应状态即可：
  1. 禁止特定两位组合（如《不要 62》）：只需记录 `bool pre6`（上一位是不是 6），当前 $d = 2$ 且 `pre6` 为真则跳过。
  2. 数位之和整除 K：把 pre 换成 `sum_mod = (sum_mod + d) % K`，到边界判 `sum_mod == 0`。
  3. 数字本身的余数（如求能被 13 整除的数）：把 pre 换成 `rem = (rem * 10 + d) % 13`。

本页要点

数位 DP 的“灵魂”永远是 `pos`，`limit`，`lead`，中间那个状态参数完全随题目需要替换！

反算抓错：怎么用 10 行暴力对拍验证？

- 数位 DP 边界极多（$A = 0$、全零、十位数进位），光靠肉眼很难看出隐蔽 Bug。
- 最稳妥的抓错法：用 10 行写一个暴力函数 `brute(L, R)`，循环 `for (int x = L; x <= R; ++x)` 直接按题意逐个检查！
- 在本地运行 `solve(R) - solve(L - 1)` 与 `brute(L, R)` 对比：
  - 先测 $[1, 1000]$ 内所有小区间；
  - 一旦发现某组 $[L, R]$ 不一致（比如 $[1, 10]$ 差了 1），马上打印出 $0 \sim 10$ 里到底多算或漏算了哪个数！

本页要点

只要小数据 $1 \sim 1000$ 与暴力结果一模一样，再配上 `long long` 和正确的 `memo` 数组大小，大概率直接 AC！

上机验证与反事实追问

3 组标准验证（以“统计 $[L, R]$ 内不含数字 4 的正整数个数”为例）

1. 最小边界：$L = 1, R = 1 \to$ 仅有 1 满足，预期输出 `1`。
2. 个位临界：$L = 1, R = 10 \to$ 去掉 4 剩下 9 个，预期输出 `9`。
3. 特殊跨位：$L = 1, R = 100 \to 100$ 个数里去掉含 4 的 19 个，预期输出 `81`。

反事实追问

如果把 `if (!limit && !lead)` 中的 `!limit` 删掉，直接对所有状态读写 `memo` 会怎样？

$\Rightarrow$ 第一次搜贴边分支时，`memo` 存下残缺计数；之后自由分支查到同一个 `memo` 会直接返回残缺答案，导致 **WA**！

本页要点

写完数位 DP 务必用 $R = 10$ 和 $R = 100$ 对比暴力结果，一秒查出边界错。

数位 DP 易错对照：WA / TLE / RE 定位

错误类型	常见错误写法	正确做法与原因
WA（漏减 1）	算成 `solve(R) - solve(L)`	把左端点 L 本身减掉了，必须算 `solve(R) - solve(L - 1)`
WA（缓存污染）	读写 `memo` 不判 `!limit`	贴上界时受当前 X 限制，不能与自由态共用缓存
WA（前导零）	没传 `lead` 或哨兵	如 05 被当成十位是 0、个位是 5 而受相邻条件误杀
WA（溢出）	返回值或 `memo` 用了 `int`	10^{18} 范围答案远超 2×10^{9}，一律用 `long long`

本页要点

数位 DP 调试四连问：减 1 了吗？判 `!limit` 了吗？防前导零了吗？开 `long long` 了吗？

例 1 | [提高] 炮兵阵地（一本通 1595）： 先读懂要求

大白话题意

给一张 $N \times M$（$N \le 100, M \le 10$）的地图，`'P'` 是平原（可放炮），`'H'` 是山地（不可放）。每门炮攻击上下左右各 2 格，要求互不攻击，求最多能放几门炮。

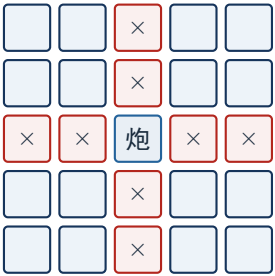
项目	说明
输入	第一行 N, M；接下来 N 行每行 M 个字符（`'P'` 或 `'H'`）
处理	满足行内左右距离 ≥ 3、行间上下距离 ≥ 3 且不放在 `'H'` 上的最大炮兵数
输出	一个整数，表示最多能摆放的炮兵部队数量
原题样例	`5 4 / PHPP / PPHH / PPPP / PHPP / PHHP` $\rightarrow$ 输出 6

本页要点

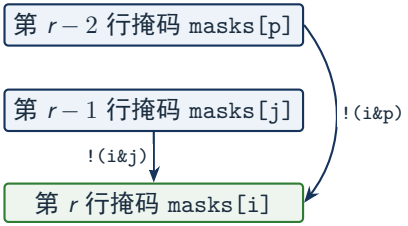
难度标签：**[提高]**。当前行同时受前 1 行和前 2 行共同约束！

例 1 | 炮兵阵地： 思路与建模图解

单门炮攻击范围（十字各 **2** 格）



按行转移：当前行依赖前两行



本页要点

设 dp[i][j] 为“当前行选掩码 i、上一行选掩码 j”时的最多炮兵数，预处理后 $k \le 60$，仅 $60^3 \times 100$ 步！

例 1 | 炮兵阵地： 样例手推跟踪表

样例：5×4 地图逐行推演（列下标 $3 \ldots 0$，H 对应掩码位 1）

行号	地图行	山地掩码 hill[row]	一种最优摆放	累计炮兵数
第 0 行	PHPP	0100（列 2 为 H）	1001（放第 0, 3 列，隔 2 格）	2
第 1 行	PPHH	0011（列 0,1 为 H）	0000（若放列 3 会打第 0 行）	$2+0=2$
第 2 行	PPPP	0000	0100（放列 2，不冲突前两行）	$2+1=3$
第 3 行	PHPP	0100	1001（距第 0 行已隔 3 行!）	$3+2=5$
第 4 行	PHHP	0110	0000 或调整第 3,4 行各放 1 门	最优合计 6

本页要点

第 3 行放列 0/3、第 4 行改放列 0 或第 1 行放列 2 都会互相牵制，只有 DP 枚举所有组合才不会漏掉最优解 6。

例 1 | 炮兵阵地： 核心步骤与模板改动

按顺序计算	纸上得到	核对问题
1. 读入山地转为 `hill[i]`	每行一个二进制掩码	`'H'` 处是否置为 1？
2. 筛行内合法 `masks[]`	$M = 10$ 时 $k \le 60$	是否同时判了左移 1 位和 2 位？
3. `dp[0][0] = 0`，其余 `-1`	标记不可达状态为 `-1`	不可达态能否错误参与转移？
4. 三重循环滚动更新 `nxt[i][j]`	满足地形和前两行不冲突	最后是否对全表取 `max`？

相对基础模板改了哪里

基础模板只记上一行 `dp[j]`；本题纵向上下攻击 2 格，所以状态多记一维变成 `dp[j][p]`（上一行 j、上上行 p），且地形检查多一个 `!(masks[i] & hill[row])`。

本页要点

记住：求最大炮兵数时，不可达状态必须设为 `-1`，不能设为 0！

例 1 | 炮兵阵地： 核心代码（1/2）：读入地形与预处理

```
int n, m; if (!(cin >> n >> m)) return 0;
vector<int> hill(n, 0);
for (int i = 0; i < n; ++i) {
    string s; cin >> s;
    for (int j = 0; j < m; ++j)
        if (s[j] == 'H') hill[i] |= (1 << j); // 山地位置标记为 1
}
vector<int> masks, cnt;
for (int s = 0; s < (1 << m); ++s)          // 左右 1、2 格内不能有炮
    if ((s & (s << 1)) == 0 && (s & (s << 2)) == 0) {
        masks.push_back(s); cnt.push_back(__builtin_popcount(s));
    }
```

本页要点

代码与 code/artillery_bitmask.cpp 一致：masks[i] & hill[row] 为 0 才不撞山。

例 1 | 炮兵阵地： 核心代码（2/2）：三维滚动转移

```
int k = (int)masks.size();
vector<vector<int>> dp(k, vector<int>(k, -1)); dp[0][0] = 0;
for (int row = 0; row < n; ++row) {
    vector<vector<int>> nxt(k, vector<int>(k, -1));
    for (int i = 0; i < k; ++i) if (!(masks[i] & hill[row]))
        for (int j = 0; j < k; ++j) if (!(masks[i] & masks[j]))
            for (int p = 0; p < k; ++p)
                if (dp[j][p] != -1 && !(masks[i] & masks[p]))
                    nxt[i][j] = max(nxt[i][j], dp[j][p] + cnt[i]);
    dp.swap(nxt);
}
int ans = 0; for (auto &r : dp) for (int v : r) ans = max(ans, v);
cout << ans << '\n';
```

本页要点

先在 i 层过滤地形、在 j 层过滤相邻行冲突，内层 p 循环被提前剪掉大量无效分支！

例 1 | 炮兵阵地： 易错与 3 组自测边界

高频易错点

- 把不可达 `dp[j][p]` 初值设为 0：凑不出来的历史状态竟然能加上 `cnt[i]`，导致答案偏大（**WA**）。
- 只判了 `s & (s << 1)`：漏了 `s & (s << 2)`，横向隔 1 格也会互相攻击（**WA**）。

3 组自测数据与预期输出

1. 全山地边界：`2 2 / HH / HH` $\rightarrow$ 预期输出 `0`。
2. 最小单格平原：`1 1 / P` $\rightarrow$ 预期输出 `1`。
3. 单行隔两格：`1 4 / PPPP` $\rightarrow$ 放在第 $1, 4$ 格，预期输出 `2`。

本页要点

自测全山地返回 0、单行 4 格返回 2，就能迅速确认地形和左右距离 2 格都写对了。

例 2 | [基础] 国王（一本通 1592）： 先读懂要求

大白话题意

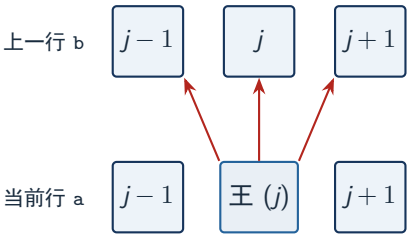
在 $n \times n$（$1 \le n \le 10$）的棋盘上放置 k（$0 \le k \le n^2$）个国王。
每个国王可以攻击周围八邻域（上、下、左、右及四个斜角），要求所有国王互不攻击，求方案总数。

项目	说明
输入	一行两个整数 n 和 k
处理	统计满足任意两个国王不相邻（含斜角）且总数为 k 的方案数
输出	一个整数，表示方案总数（无法放置则输出 0）
原题样例	样例 1：3 2 $\rightarrow$ 16；样例 2：4 4 $\rightarrow$ 79

本页要点

难度标签：[基础]。只需要记当前行状态和已放置国王数 c，且只依赖上一行！

例 2 | 国王： 思路与建模图解



三种行间冲突均禁止：

1. 正上方：`(a & b) != 0`
2. 左上方：`(a & (b << 1)) != 0`
3. 右上方：`(a & (b >> 1)) != 0`

本页要点

把“上一行 b 左移 1 位”和“右移 1 位”，就刚好把斜向八邻域的冲突也变成了按位与！

例 2 | 国王：样例手推跟踪表（$n=3, k=2$）

样例：3×3 棋盘放 2 个国王（行内合法掩码为 000，001，010，100，101）

阶段	当前放法举例	已放 c	合法方案数
初始	虚拟第 0 行空态 dp[0][0] = 1	0	1
第 1 行	放 0 个有 1 种；放 1 个有 3 种；放 2 个（101）有 1 种	$0\sim 2$	$c=2$ 有 1 种
第 2 行	若上行 001，本行禁 001/010，可选 000/100	$0\sim 2$	累加兼容态
第 3 行	继续向后转移，汇总 dp[i][2]	2	最终得 16

本页要点

反算验证：9 格选 2 格共 $\binom{9}{2}=36$ 种，减去相邻冲突的 20 对，恰好剩 16 种！

例 2 | 国王：核心步骤与模板改动

按顺序计算	纸上得到	核对问题
1. 筛行内不相邻 `masks[]`	同行左右无相邻 1	数组长度是 n 还是 m？
2. `dp[0][0] = 1`（`long long`）	0 行放 0 个国王方案为 1	方案数很大时有没有爆 `int`？
3. 判三向行间冲突	`a&b`, `a&(b<<1)`, `a&(b>>1)`	有没有漏掉斜向冲突？
4. 累加 `nxt[i][c] += dp[j][c-cnt[i]]`	枚举总国王数 c 从 `cnt[i]` 到 `need`	下界 `cnt[i]` 防止负数越界？

相对基础模板改了哪里

基础模板只有一维 `dp[i]`；本题要求总国王数恰好为 k，因此在状态中增加一维“当前累计国王数 c”，变成 `dp[i][c]`。

本页要点

循环顺序：`row` → `i` → `j` → `c`，内层只把合法的 c 加上 `cnt[i]`。

例 2 | 国王： 核心代码（1/2）：预处理与状态定义

```cpp
int n, need; if (!(cin >> n >> need)) return 0;
vector<int> masks, cnt;
for (int s = 0; s < (1 << n); ++s) {       // 筛出左右不相邻的合法单行掩码
    if ((s & (s << 1)) == 0) {
        masks.push_back(s);
        cnt.push_back(__builtin_popcount(s));
    }
}
int m = (int)masks.size();
vector<vector<long long>> dp(m, vector<long long>(need + 1, 0));
dp[0][0] = 1;                              // masks[0] == 0，放 0 个国王有 1 种空态
```

本页要点

代码与 code/king_bitmask.cpp 一致：方案总数在 $n = 10$ 时可达 10^{12} 以上，必须开 long long！

例 2 | 国王： 核心代码（2/2）：八邻域冲突判断与累加

```
for (int row = 0; row < n; ++row) {
    vector<vector<long long>> nxt(m, vector<long long>(need + 1, 0));
    for (int i = 0; i < m; ++i)
        for (int j = 0; j < m; ++j) {
            int a = masks[i], b = masks[j];
            if ((a & b) || (a & (b << 1)) || (a & (b >> 1))) continue;
            for (int c = cnt[i]; c <= need; ++c)
                nxt[i][c] += dp[j][c - cnt[i]];
        }
    dp.swap(nxt);
}
long long ans = 0; for (int i = 0; i < m; ++i) ans += dp[i][need];
cout << ans << '\n';
```

本页要点

注意 c 从 cnt[i] 开始循环，保证 c - cnt[i] >= 0 不会越界。

例 2 | 国王： 易错与 3 组自测边界

高频易错点

- 方案数用 `int` 存储：$n = 10, k = 15$ 时答案早已超过 $2^{31} - 1$，直接溢出变负数（**WA**）。
- 漏判 `a & (b >> 1)`：只判了一边斜角，导致右上-左下斜相邻未被禁止（**WA**）。

3 组自测数据与预期输出

1. 放 0 个国王：`3 0` $\rightarrow$ 预期输出 1（全空棋盘 1 种）。
2. 原题样例 2：`4 4` $\rightarrow$ 预期输出 79。
3. 无法放置过多国王：`2 2` $\rightarrow 2 \times 2$ 最多放 1 个国王，预期输出 0。

本页要点

用 `3 0` 测出 1、用 `2 2` 测出 0、用 `4 4` 测出 79，三组通过就稳了！

例 3 | [提高] Windy 数（一本通 1587）： 先读懂要求

大白话题意

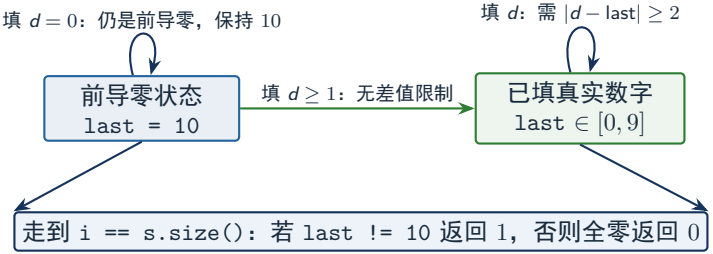
不含前导零、且相邻两个数位之差至少为 2 的正整数称为 Windy 数（所有 $1 \sim 9$ 都算）。给定 A, B（$1 \le A \le B \le 2 \times 10^9$），求 $[A, B]$ 内有多少个 Windy 数。

项目	说明
输入	一行两个整数 A, B
处理	转化为 $\text{solve}(B) - \text{solve}(A-1)$，从高位到低位记忆化搜索
输出	一个整数，表示 $[A, B]$ 内 Windy 数的个数
原题样例	样例 1：1 10 $\to$ 9；样例 2：25 50 $\to$ 20

本页要点

难度标签：**[提高]**。同时考验“上界贴合 `tight`”和“前导零哨兵 `last = 10`”两大技巧！

例 3 | Windy 数： 思路与建模图解



本页要点

巧妙设计：用 `last = 10` 这个不存在的“数字”兼任前导零标记，省掉一个 `bool lead` 参数！

例 3 | Windy 数： 样例 1 手推（1 10）

样例 1：solve(10) - solve(0) 的推演

函数调用	十位 d	个位 d	是否合法	累计计数
solve(10)	0（last=10）	0（全零态）	边界 last==10 → 返 0	0
solve(10)	0（last=10）	$1 \sim 9$	无前驱限制，全部合法	9
solve(10)	1（last=1，tight=1）	只能填 0	$\lvert 0-1\rvert = 1 < 2$（非法！）	0
solve(0)	—	—	last==10 → 返 0	0
总结果	—	—	solve(10) - solve(0)	$9-0=9$

本页要点

为什么 10 不算 Windy 数？因为 $|0-1| = 1 < 2$；而 $1 \sim 9$ 都算，所以 $[1, 10]$ 共有 9 个！

例 3 | Windy 数： 样例 2 手推（25 50）

样例 2：直接按十位分类手数 $[25, 50]$ 内的 Windy 数，核对是否等于 20

十位数字	允许的个位（$\lvert d - \text{十位} \rvert \geq 2$ 且在 $[25, 50]$ 内）	个数	说明
2（25 . . . 29）	$5, 6, 7, 8, 9$	5	$\lvert d - 2 \rvert \geq 2 \implies d \geq 4$
3（30 . . . 39）	$0, 1, 5, 6, 7, 8, 9$	7	跳过 $2, 3, 4$
4（40 . . . 49）	$0, 1, 2, 6, 7, 8, 9$	7	跳过 $3, 4, 5$
5（仅 50）	0	1	$\lvert 0 - 5 \rvert = 5 \geq 2$ 合法
总计	—	**20**	与 `solve(50)-solve(24)` 一致

本页要点

手推 $5 + 7 + 7 + 1 = 20$，和原题样例 2 输出完全吻合！

例 3 | Windy 数： 核心步骤与模板改动

按顺序计算	纸上得到	核对问题
1. `solve(x)` 转字符串 `s`	得到十进制各位上界	`memset(memo, -1)` 做了吗？
2. 判断边界 `i == s.size()`	返回 `last != 10`	全 0 会不会被多算成 1？
3. 检查 `!tight && memo != -1`	自由态直接复用缓存	为什么不能省略 `!tight`？
4. 枚举 $d = 0 \ldots \text{cap}$ 递归	`solve(b) - solve(a - 1)`	$a-1$ 会不会漏减 1？

相对基础模板改了哪里

将 `lead` 与 `pre` 合并成 `last`（取 $0 \sim 9$ 或哨兵 10），判断规则为 `last != 10 && abs(d - last) < 2`。

本页要点

缓存条件写成 `!tight`（自由态），因为在 `last == 10` 时也只有全零路径会到达。

例 3 | Windy 数： 核心代码（1/2）：dfs 边界与读缓存

```
long long solve(long long x) {
    string s = to_string(x);
    long long memo[20][11]; memset(memo, -1, sizeof(memo));
    auto dfs = [&](auto &&self, int i, int last, bool tight) -> long long {
        if (i == (int)s.size()) return last != 10; // 填过非零数才算正整数
        if (!tight && memo[i][last] != -1)
            return memo[i][last];          // 自由态才可复用备忘录
        int cap = tight ? (s[i] - '0') : 9;
        long long ans = 0;
```

本页要点

代码与 code/windy_digit.cpp 一致：使用 C++17 泛型 Lambda auto &&self 递归简洁明了。

例 3 | Windy 数： 核心代码（2/2）：枚举数位与差分求解

```
        for (int d = 0; d <= cap; ++d) {
            if (last != 10 && abs(d - last) < 2) continue; // 相邻差>=2
            int nxt = (last == 10 && d == 0) ? 10 : d;     // 前导零处理
            ans += self(self, i + 1, nxt, tight && (d == cap));
        }
        return tight ? ans : (memo[i][last] = ans);
    };
    return dfs(dfs, 0, 10, true);
}
int main() {
    long long a, b; if (cin >> a >> b) cout << solve(b) - solve(a - 1) << '\n';
}
```

本页要点

因 $A \geq 1$，$A - 1 \geq 0$，to_string(0) 返回 "0"，无需额外特判负数。

例 3 | Windy 数： 易错与 3 组自测边界

高频易错点

- 写错边界返回 `return 1`：把全零数字 0 也算成了 Windy 数（虽然差分 $solve(B) - solve(A-1)$ 时可能抵消，但单独看 `solve(x)` 会多 1，概念不清）。
- 差值条件写成 `<= 2`：题目要求“至少为 2”，即 ≥ 2，所以跳过条件是 `< 2`。

3 组自测数据与预期输出

1. 最小单点：`1 1` → 预期输出 `1`（仅数字 1）。
2. 原题样例 1：`1 10` → 预期输出 `9`（$1 \sim 9$）。
3. 原题样例 2：`25 50` → 预期输出 `20`。

本页要点

用 `1 1`、`1 10`、`25 50` 跑一遍，既能验哨兵又能验 `tight` 上界！

常见错误对照表（题号 | 常见错误 | 正确做法）

题目 / 场景	常见错误写法	正确做法与原因
炮兵阵地（1595）	忘记 dp 不可达态设 -1	不可达初态若为 0 会被加上 cnt[i]，必须设 -1
炮兵阵地（1595）	写 s & (s << 1) == 0	== 优先级高于 &，必须写 (s & (s << 1)) == 0
国王（1592）	方案数用 int 存储	$n = 10$ 时方案数超 10^{12}，必须用 long long
Windy 数（1587）	读写 memo 没判 !tight	贴上界与自由态不同，只对 !tight && !lead 缓存
Windy 数（1587）	区间减法写 solve(b) - solve(a)	把左端点 a 减没了，必须算 solve(b) - solve(a - 1)

本页要点

状压查“括号与 -1”，数位查“!tight 与减 1”，十之八九的 WA 都能秒定位。

评测结果定位指南：CE / WA / RE / TLE

评测结果	本章快速定位思路
CE（编译错误）	检查是否使用了 C++17 递归 Lambda（需开 `-std=c++17`），或 `memset` 漏了头文件 `<cstring>`。
WA（答案错误）	90% 是位运算没套括号、dp 初值未设 `-1`、数位 `memo` 忘了判 `!tight`、或前导零误杀一位数。
RE（运行错误）	状压 `1 << n` 超过数组大小，或国王计数 `c - cnt[i]` 减成负数下标越界。
TLE（超时）	状压内层直接跑 $0 \ldots 2^m - 1$ 没先预处理 `ok[]`；或数位 DP 忘了写 `memo` 缓存。

本页要点

遇到 WA 先拿 $N = 1, M = 1$ 或 $A = 1, B = 10$ 手推对比，不要盲目猜代码。

提交前检查清单（六必查）

1. 所有涉及 `&`、`|` 的判断条件（如 `(s & hill) == 0`），是否都加了外层小括号？
2. 状压转移前是否先预处理合法行掩码 `masks[]`？求最值时不可达状态是否标为 `-1`？
3. 数位 DP 的 `memo` 读写前，是否严格加了 `if (!limit && !lead)`（或 `!tight`）？
4. 前导零阶段（如 00035）有没有用 `lead` 或哨兵 `last = 10` 隔离相邻规则？
5. 区间 $[A, B]$ 查询调用 `solve(B) - solve(A - 1)` 时，左端点 A 是否已减 1 且防负数？
6. 计数范围高达 $10^{12} \sim 10^{18}$ 时，返回值和 `memo`、`dp` 是否均用了 `long long`？

本页要点

提交前把这 6 条过一遍，能直接挡住绝大多数低级失误。

本章全部练习与完成标准

题目（一本通题号）	难度标签	练习目标与配套源码
学校 OJ：1595	炮兵阵地	[提高]	预处理行掩码 + 滚动二维状压，code/artillery_bitmask.cpp
学校 OJ：1592	国王	[基础]	八邻域位运算冲突 + 计数状压，code/king_bitmask.cpp
学校 OJ：1584	骑士	[提高]	日字形攻击跨两行冲突，巩固双行状压
学校 OJ：1587	Windy 数	[提高]	前导零哨兵 + tight 记忆化，code/windy_digit.cpp
学校 OJ：1589	不要 62	[基础]	禁用 4 与 62，入门数位 DP 必刷
学校 OJ：1591	数字计数	[提高]	统计 $[a, b]$ 中 $0 \ldots 9$ 出现次数，练透 lead 处理

完成标准

独立写出位运算判冲突与预处理掩码；独立写出带 limit 与 lead 的数位 DP；AC 上述至少 4 道题。

课后安排与下一章预告

- 复习：合上课件再推一遍为什么 $M = 10$ 预处理完《炮兵阵地》合法掩码只剩 60 个。
- 练习：独立复写 code/ 下三份源码，并在《Windy 数》中试着删掉前导零哨兵观察 $1 \sim 9$ 的输出变化。
- 下一章预告：第 5 章《单调队列优化 DP》——当 DP 转移长成 $dp[i] = w(i) + \min_{i-m \le j < i} g(j)$ 每次要扫一段窗口最值时，怎么用双端队列把 $O(nm)$ 降维成均摊 $O(n)$！

本页要点

下一章从“把状态压进二进制/数位”转向“把重复扫描的窗口压缩进单调队列”。

本章一句话

本章一句话

小棋盘把一整行压成整数位掩码，大整数区间把数字拆成数位带着上界记忆化向前搜。

谢谢！