博弈论与 SG 函数

必胜态与必败态 + 巴什与对称 + Nim 异或 + SG 定理

HGNU-ACM 实验室算法组（全系列收官之章）

承接第 4 章计数与期望：用状态倒推、异或平衡与 mex 判定谁赢

配套：3 道真题精讲（一本通 1663 / 1664 / 1665）+ 全系列 8 篇总览回顾

本章目录（1/2）

公平组合游戏（ICG）与必胜态 N / 必败态 P

巴什博弈与威佐夫/对称博弈直觉

Nim 博弈与 Bouton 异或和定理

第 1–3 节打通组合博弈的核心直觉：从生活拿糖果游戏导出 P/N 态倒推，再到巴什取余与 Nim 异或平衡。

本页要点

不背结论，先手算小例子，看清“怎么把必败留给对手”。

本章目录（2/2）

有向图 Sprague-Grundy（SG）函数与 mex 运算

配套例题精讲：取石子 1、取石子 2 与移棋子

常见错误、检查清单与全系列收官

第 4–6 节掌握万能的有向图 SG 函数与 mex，精讲 3 道一本通真题，并为 8 篇全系列圆满收官。

本页要点

学会 SG 定理，就能把任意多个独立有向图游戏统一成 Nim 异或。

先玩一个小游戏：桌上 4 颗糖，每次拿 1 或 2 颗

生活场景

第 4 章我们算过“随机发生这件事的概率”，这一章换个问法：如果双方都绝顶聪明、没有随机性，谁一定能赢？

- 规则：桌上最初有 $n = 4$ 颗糖；你先拿，每次只能拿 1 或 2 颗；拿到最后一颗的人获胜。
- 先手拿 1 颗，剩 3 颗：无论对手拿 1（剩 2）还是拿 2（剩 1），你都能一口气拿完剩下的！
- 先手拿 2 颗，剩 2 颗：对手直接拿光剩下的 2 颗，你输！
- 动笔想想：为什么 $n = 4$ 时先手拿 1 颗稳赢，而 $n = 3$ 时怎么拿都会输？

本页要点

博弈论的第一步不是背公式，而是从终点 0 倒着推：哪几个数字是“谁遇到谁输”？

为什么需要状态倒推？暴力搜博弈树会怎样

- 有人想：写个递归函数 `dfs(n)`，枚举先手拿 1 还是拿 2，再让对手递归……
- 听起来很自然，但我们算一笔操作次数账：每次有 2 个分支，深度最多为 n。
- 没有记忆化时，最坏访问节点数约为 $O(2^n)$！即使 $n = 50$，$2^{50} \approx 1.1 \times 10^{15}$，比赛 1 秒限制（约 10^8 次操作）必 **TLE**。
- 原因在于大量重复计算：比如“剩 2 颗”会被 $4 \to 2$、$4 \to 3 \to 2$ 反复搜到无数次。

核心想法

总共只有 $0, 1, \ldots, n$ 这 $n+1$ 个不同局面！只要从终点 0 开始从小到大倒推，每个局面只算一次。

本页要点

博弈状态只看“当前还剩多少”，不看怎么走到这里，所以从终点倒推就能把指数级压成线性！

新术语大白话：公平组合游戏、P 态与 N 态

- 公平组合游戏（**ICG**）：两人轮流走、没有运气成分（不掷骰子）、两人能做的操作规则完全对称、不能走的一方输，且保证有限步内结束。
- **P 态**（**Previous win**，先手必败态）：轮到当前玩家走时，他无路可走，或者不管怎么走都会送对手进入必胜态（所以“上一位走完的人”赢）。
- **N 态**（**Next win**，先手必胜态）：轮到当前玩家走时，只要存在哪怕一条路能走到 **P 态**，他就走这条路，把必败留给对手！

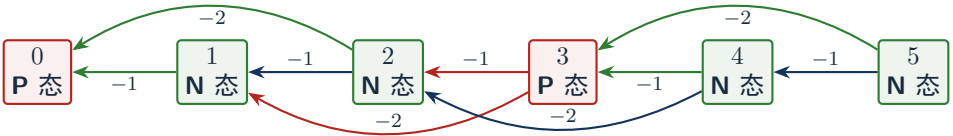
记法口诀

站在当前轮到行动的人视角：**N = Now win**（现在的人赢），**P = Lose now**（现在的人输，上一个行动的人赢）。

本页要点

判断一个局面是 P 还是 N，只看它能一步走到的所有后继局面里有没有 P。

状态转移图解：糖果游戏 $0 \sim 5$ 颗的有向转移



- 绿箭头表示抢到 **P** 态的制胜步（如 $4 \to 3$、$5 \to 3$）；红箭头表示被迫送对手去 **N** 态（如 $3 \to 2$ 与 $3 \to 1$）。
- 状态 2 虽然也能走到 1（N 态），但只要它有一条绿箭头 $2 \to 0$（P 态），2 就是 N 态！

本页要点

N 态只需要“有一条绿色出边指到 P 态”；P 态则是“所有出边都是红色，全指到 N 态”。

从终点倒推手算表（每次拿 1 或 2 颗，$n = 0 \sim 5$）

当前糖数 x	可走到的后继	后继状态标签	当前判定	原因与最优走法
0	无（空集 $\varnothing$）	无	**P 态**	轮到你时已无糖可拿，直接判负
1	$\{0\}$	$\{P\}$	**N 态**	拿 1 颗走到 0（P 态）即获胜
2	$\{1, 0\}$	$\{N, P\}$	**N 态**	拿 2 颗走到 0（P 态）即获胜
3	$\{2, 1\}$	$\{N, N\}$	**P 态**	拿 1 或 2 都只能送到 N 态，必败
4	$\{3, 2\}$	$\{P, N\}$	**N 态**	存在后继 3 为 P 态，先拿 1 颗
5	$\{4, 3\}$	$\{N, P\}$	**N 态**	存在后继 3 为 P 态，先拿 2 颗

- 注意第 3 行与第 5 行：即使部分后继是 N 态，只要列表里出现哪怕一个 P 态，当前就立刻判定为 N 态！

本页要点

手算倒推表时，每行只检查“后继状态标签里有没有 P”，有则 N，全 N 则 P。

为什么这样倒推一定对？两条公理与三步验证

- 为什么不会出现“两个状态互相推来推去，算不出谁输谁赢”？因为 ICG 保证有限步内结束，状态图是有向无环图（**DAG**），每一步都在向终点靠近。

验证阶段	具体内容（以“按拓扑逆序/从小到大倒推”为例）
初始化	终点状态（无后继）记为 `win[0] = false`（P 态），显然无路可走必败
保持	算到 x 时，所有后继 $y < x$ 的胜负已确定；存在 `win[y]==false` 则 `true`，否则 `false`
终止	算完目标状态 n 后，`win[n]` 即为最初先手在双方最优策略下的真实胜负

- 只要满足无环性，从小到大或按拓扑逆序就能一次算清所有状态的 P/N 归属。

本页要点

两条公理之所以严谨成立，靠的是“局面单向推进、绝不走回头路”的有向无环结构。

模板代码：通用一维状态倒推打表（拿 1 或 2 颗）

```cpp
int main() {
    int n; cin >> n;                                    // 读入初始糖果总数 n
    vector<bool> win(n + 1, false);                     // 初值全 false，终点 win[0] =
    false
    for (int i = 1; i <= n; ++i) {                      // 从小到大倒推每个局面的胜负
        if (i >= 1 && !win[i - 1]) win[i] = true; // 拿 1 颗能送到 P 态，则当前是 N
    态
        if (i >= 2 && !win[i - 2]) win[i] = true; // 拿 2 颗能送到 P 态，则当前是 N
    态
    }
    cout << (win[n] ? "N(win)" : "P(lose)") << '\n'; // 输出先手是必胜还是必败
    return 0;
}
```

本页要点

初值 false 表示默认无必胜走法（且 win[0]=false 必败），一旦能走到 false 就改为 true！

上机验证与反事实追问

测试类型	输入 n	预期输出	手算核对说明
普通情况	5	`N(win)`	拿 2 颗剩 3（P 态），先手必胜
最小边界	0	`P(lose)`	一开始就无糖可拿，先手直接输
特殊必败	3	`P(lose)`	拿 1 剩 2、拿 2 剩 1，全送给对手 N 态

反事实追问

若把第 3 行初值误写成 `vector<bool> win(n + 1, true)`，输入 $n = 1$ 会怎样？
答案：`win[0]` 变成 `true` 后，`!win[0]` 为假，后面全判定不到 `false`，程序完全失灵！

本页要点

写完倒推代码后，先测 $n = 0$ 和第一个必败态 $n = 3$，能立刻抓出初值或边界错误。

ICG 与 NP 态易错对照：什么时候要升级方法？

常见错误场景	错误写法	正确做法与评测后果
`win[0]` 初值设反	`win[0] = true`（误以为剩 0 算赢）	面对 0 颗时无路可走，必是 `false`（WA）
忘记判断数组越界	`i >= 2` 漏写，直接访问 `win[i - 2]`	当 $i = 1$ 时访问 `win[-1]` 越界（RE / 乱值 WA）
大范围硬跑一维倒推	$n = 10^9$ 仍开数组 `win(n+1)`	空间超限（MLE）且时间 $O(n)$ 超时（TLE）

本页要点

倒推表适合 n 较小的单游戏；大 n 需用下节巴什取余，多堆/多棋子需用第 3–4 节 Nim 与 SG！

从 $1 \sim 2$ 颗推广到 $1 \sim k$ 颗：先手算找周期

新问题

一堆共有 n 颗石子，两人轮流取，每次最少拿 1 颗、最多拿 k 颗，拿到最后一颗的人赢。

- 先固定 $k = 3$（每次最多拿 3 颗），用上一节“倒推”方法手算 $n = 0 \sim 8$ 的 P/N 态：
- $n = 0$：**P**；$n = 1, 2, 3$：一口气拿光就赢，全都是 **N**；
- $n = 4$：拿 $1, 2, 3$ 分别剩 $3, 2, 1$（全是 N 态），所以 $n = 4$ 是 **P**！
- $n = 5, 6, 7$：分别拿 $1, 2, 3$ 颗就能留给对手 4（P 态），所以全是 **N**；
- $n = 8$：拿 $1, 2, 3$ 分别剩 $7, 6, 5$（全是 N 态），所以 $n = 8$ 又是 **P**！

本页要点

敏锐的同学已经发现了：当 $k = 3$ 时，每 4 个数出现一次 P 态（$0, 4, 8, 12, \ldots$）！

为什么不能只靠循环打表？算一笔复杂度账

- 上一节的倒推循环：对每个 $i \in [1, n]$，要枚举拿 $1 \sim k$ 颗，总时间复杂度是 $O(nk)$。
- 看看真题数据范围：如果 $n = 10^5, k = 10^5$（或者更常见的 $n \le 10^9$），$O(nk)$ 的循环需要执行高达 10^{10} 次！
- 10^{10} 次运算远超竞赛 1 秒 10^8 次的上限，必然触发 **TLE**。
- 而且哪怕用一维数组记 `win[n]`，$n = 10^9$ 也会直接 **MLE**（内存超限）。

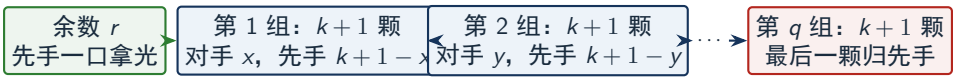
核心想法

既然 P 态整齐地隔着 $k+1$ 出现，我们只需要一步取余 `n % (k + 1)`，就能在 $O(1)$ 时间直接出答案！

本页要点

找规律把 $O(nk)$ 降到 $O(1)$，是组合博弈从“模拟倒推”跨入“数学必胜构造”的第一步。

巴什博弈核心图解：分组凑 $(k+1)$ 配对策略



- 把 n 写成带余除法：$n = q(k+1) + r$，其中 $0 \le r \le k$。
- 若 $r \neq 0$：先手第一步直接拿走 r 颗！此后每当对手拿 x 颗（$1 \le x \le k$），先手就稳稳拿 $(k+1)-x$ 颗，凑满一整组 $k+1$ 颗，最终一定是先手拿走最后一颗！
- 若 $r = 0$：先手没有多余的 r 可拿，反而是后手掌握“凑 $k+1$”的主动权，先手必败！

本页要点

巴什博弈的本质是“凑数配对”：谁把 $k+1$ 的倍数留给对手，谁就永远掌握配对主动权！

巴什博弈手算跟踪表（$n = 11, k = 3$，每组 $k + 1 = 4$）

回合	行动方	操作前	拿走颗数	操作后	模 4 余数与说明
第 1 步	先手	11	3（$= 11 \bmod 4$）	8	余数为 0，成功把 P 态留给后手
第 2 步	后手	8	假设拿 1	7	余数为 3，先手只需凑 $4 - 1 = 3$
第 3 步	先手	7	3	4	余数重回 0，先手稳控局面
第 4 步	后手	4	假设拿 2	2	余数为 2，先手只需凑 $4 - 2 = 2$
第 5 步	先手	2	2	0	剩余为 0，先手拿完最后一颗获胜！

- 无论后手在第 2 步拿 $1, 2, 3$ 中的哪一个，先手总能拿 $4 - x \in \{1, 2, 3\}$，合法且必将余数拉回 0！

本页要点

跟踪表清楚展示了：先手每次操作后，剩余石子数一定是 4 的倍数，直到变成 0。

为什么巴什取余判胜负一定对？不变量三步检验

- 设关键不变量为：“每轮先手拿完后，剩余石子数 rem 始终满足 $rem \equiv 0 \pmod{k+1}$”。

阶段	为什么一定成立（以 $n \bmod (k+1) \neq 0$ 时先手为例）
初始化	第一步拿走 $r = n \bmod (k+1)$ 颗（$1 \le r \le k$ 合法），剩余 $n-r$ 是 $k+1$ 的倍数
保持	对手拿 $x \in [1,k]$ 后，剩余 $rem - x$；先手拿 $y = k+1-x \in [1,k]$，新剩余仍是倍数
终止	每次两人共消去 $k+1$ 颗，有限步后必降到 0；因总是先手使剩余变 0，先手必拿最后一颗

- 反过来，若最初 $n \equiv 0 \pmod{k+1}$，则先手第一步就会打破倍数，后手接管上述不变量而稳赢！

本页要点

不用死记“$n \bmod (k+1) \neq 0$”，记住“先手能把倍数留给对手，不变量就锁死在先手手里”。

模板代码：巴什博弈 $O(1)$ 判定

```cpp
int main() {
    ios::sync_with_stdio(false); cin.tie(nullptr); // 加速标准输入输出
    long long n, k;                                // 用 long long 防止大范围溢出
    if (!(cin >> n >> k)) return 0;                // 读入石子总数 n 和单次上限 k
    if (n % (k + 1) != 0) {                        // 余数非 0，先手可先取走余数
        cout << "First Win\n";                     // 构造 k+1 倍数留给对手，先手必
    胜
    } else {                                       // 恰好是 k+1 的倍数，先手无余数
    可取
        cout << "Second Win\n";                    // 对手反过来凑 k+1，后手必胜
    }
    return 0;
}
```

本页要点

为什么写 `n % (k + 1)`？因为一轮两人凑齐 $k+1$ 颗；且 `k` 很大时需用 `long long` 防溢出！

上机验证与反事实追问

测试类型	输入 (n, k)	预期输出	手算核对说明
普通情况	`23 3`	`First Win`	$23 \bmod 4 = 3 \neq 0$，先手拿 3 颗剩 20
最小边界	`1 1`	`First Win`	$1 \bmod 2 = 1 \neq 0$，先手拿 1 颗即赢
必败边界	`4 3`	`Second Win`	$4 \bmod 4 = 0$，先手拿 $1/2/3$ 都被后手清零

反事实追问

若改成拿到最后一颗的人反而输，必胜条件还是 `n % (k + 1) != 0` 吗？
答案：不是！此时要把最后 1 颗逼给对手，必败点变为 $1, k+2, \ldots$，判定式变为 `(n - 1) % (k + 1) != 0`。

本页要点

换了“拿最后 1 颗算输”时，目标从留 0 变成留 1，判定式立刻变成 `(n - 1) % (k + 1) != 0`。

拓展直觉：对称博弈与威佐夫（Wythoff）博弈

- 对称模仿策略：圆形空桌轮流放等大硬币（不能重叠），先手占圆心，此后模仿对手做中心对称即可必胜！
- 威佐夫博弈：两堆石子 (a, b)，每次可从一堆任取，或从两堆同时拿走相同个数。不能拿者输。

博弈模型	允许操作	必败态（P 态）判定规律
巴什博弈（1 堆）	拿 $1 \sim k$ 颗	$n \equiv 0 \pmod{k+1}$
两堆对称（无同时取）	任选一堆拿任意正数颗	$a = b$（后手模仿先手在另一堆拿等量）
威佐夫博弈（可同时取）	一堆任取，或两堆同取	奇异局势 $(0,0), (1,2), (3,5), \ldots$，满足 $a = \lfloor (b-a) \cdot \frac{\sqrt{5}+1}{2} \rfloor$

本页要点

对称博弈的核心是“模仿对手的对称动作”；两堆若允许同时取等量，P 态则与黄金分割比挂钩。

巴什与对称博弈易错对照

常见错误场景	错误写法	正确做法与评测后果
模数漏加 1	写成 `n % k != 0`	每轮配对和是 $k+1$，漏加 1 会把 $(4,3)$ 判错（WA）
忽略“至少拿 1 颗”	认为 $k \geq n$ 时结果不确定	当 $k \geq n$ 时 $n < k+1$，故 $n \bmod (k+1) = n \neq 0$，先手一把拿完必胜
输出格式不审题	一本通 1663 输出 `win/lose`	1663 要求输出 1 或 2，1664 才是 `win/lose`（WA）
威佐夫精度误差	`double` 乘黄金比对大数丢精度	数据大时需用 `long double` 避免浮点舍入误差（WA）

本页要点

遇到单堆限额取石子，先看是“拿光赢”还是“拿光输”，再核对模 $k+1$ 与输出格式。

多堆石子引入：从两堆对称到三堆 $(1, 2, 3)$

Nim 游戏规则

桌上摆着 n 堆石子，数量分别为 $a_1, a_2, \ldots, a_n$。两人轮流选任意一堆，从中拿走至少 1 颗（可拿光该堆），无石子可拿者输。

- 若只有 1 堆：先手直接全拿光，必胜！
- 若有 2 堆 (a, b)：当 $a = b$（如 $(3, 3)$），后手永远“模仿先手在另一堆拿相同颗数”，先手必败；当 $a \neq b$，先手把多的那堆砍成和少的相等，先手必胜！
- 那 3 堆 $(1, 2, 3)$ 呢？试试先手拿哪一堆：拿成 $(0, 2, 3)$、$(1, 1, 3)$、$(1, 2, 2)$……怎么都像送分？

本页要点

两堆靠“相等”配对就能看出来；三堆以上 $(1, 2, 3)$ 也是先手必败，背后的规律正是二进制异或！

为什么需要 Bouton 定理？暴力搜各堆为什么会 TLE

- 模仿第 1 节写个记忆化搜索 `dfs(a1, a2, ..., an)` 行不行？
- 看看一本通 1664 的数据范围：堆数 $n \le 5 \times 10^4$，每堆石子数 $X_i \le 10^5$！
- 即使只有 4 堆且每堆 1000 颗，总状态数也有 $(1001)^4 \approx 10^{12}$，内存和时间全部炸裂！
- 当 $n = 5 \times 10^4$ 时，状态维数直接变成 5 万维，暴力搜索连数组都开不出来，必 **TLE / MLE**。

核心想法

不需要记录每堆剩多少！1901 年数学家 Bouton 证明：只需把所有堆的石子数做按位异或（**XOR**），用 $O(n)$ 时间就能一锤定音。

本页要点

面对高维多堆状态，必须找一个“一维不变量”来代替整张状态表，这个不变量就是异或和！

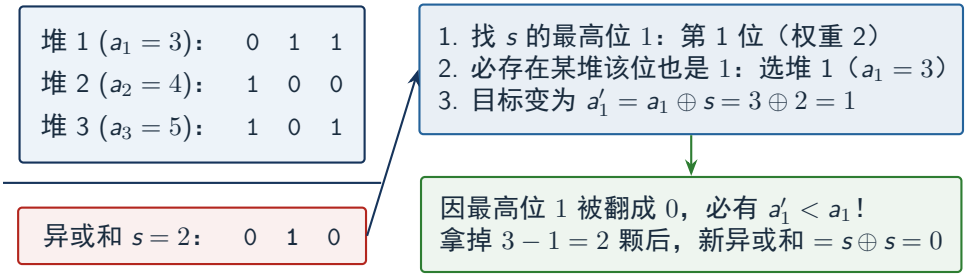
Bouton 异或和定理：把石子拆成“二进制 2 的幂”

- 按位异或 $\oplus$：把数字写成二进制，同一位上“相同得 0，不同得 1”。换句话说，这一位上 1 的个数是偶数，异或结果就是 0；是奇数，结果就是 1。
- **Bouton 定理**：在标准 Nim 游戏中，设全体异或和 $s = a_1 \oplus a_2 \oplus \cdots \oplus a_n$：
  - $s \neq 0 \iff$ 先手必胜（**N** 态）；
  - $s = 0 \iff$ 先手必败（**P** 态）。
- 回头看 $(1, 2, 3)$：$1 = 01_2$，$2 = 10_2$，$3 = 11_2$。每一位上都有两个 1（偶数个），所以 $1 \oplus 2 \oplus 3 = 0$，先手必败！

本页要点

异或和 $s = 0$ 的直觉是“二进制每一位上的 1 都是成双成对出现的，处于完全平衡态”。

TikZ 位对齐图解：为什么 $s \neq 0$ 总能一步消回 0？



- 关键在于最高位 1：a_1' 在该位从 1 变成 0，而比它更高的位全都不变（s 更高位全为 0），所以不管低位怎么变，新数 a_1' 一定严格小于 a_1，合法可取！

本页要点

构造必胜步的三步口诀：算总异或 $s \to$ 找最高位 $1 \to$ 挑含该位的堆改成 $a_i \oplus s$！

Nim 博弈手算模拟表：$(3,4,5)$ 的完整对局

回合	行动方	当前各堆	异或和 s	操作与说明
初始	—	$(3,4,5)$	2	非 0，先手处于 N 态
第 1 步	先手	$(1,4,5)$	0	堆 1 拿 2 颗（$3\oplus 2=1$），锁死平衡
第 2 步	后手	$(1,4,2)$	7	假设后手堆 3 拿 3 颗，平衡被打破
第 3 步	先手	$(1,3,2)$	0	堆 2 变 $4\oplus 7=3$，异或重归 0
第 4 步	后手	$(1,3,0)$	2	假设后手拿光堆 3，s 又变非 0
第 5 步	先手	$(1,1,0)$	0	堆 2 拿 2 颗，此后模仿拿完必胜！

- 注意第 3 步的演算：后手剩 $(1,4,2)$ 时 $s=7$，最高位 4 在堆 2，故把 4 改成 $4\oplus 7=3$，立刻回到 $(1,3,2)$ 异或为 0！

本页要点

先手每次都只需要“消去总异或和的最高位 1”，就能永远把异或和 0 踢给对手。

为什么 Bouton 定理一定对？三步严密证明

验证项目	证明逻辑（把全体局面分成 $S_0=\{s=0\}$ 与 $S_1=\{s\neq 0\}$ 两类）
1. 终点归属	全空局面 $(0,0,\ldots,0)$ 的异或和为 0，恰好落在 S_0（P 态）
2. S_0 必破	从 $s=0$ 任选一堆 a_i 减为 $a_i'<a_i$，新异或和 $s'=a_i\oplus a_i'\neq 0$，必入 S_1
3. S_1 可回	从 $s\neq 0$ 挑含 s 最高位 1 的堆 a_i，令 $a_i'=a_i\oplus s<a_i$，新异或和必重归 S_0

- 这三条恰好完整对应第 1 节的两条公理：没有出边能留在 S_0，而 S_1 总有出边回到 S_0！
- 因此 S_0 就是全部 P 态的集合，S_1 就是全部 N 态的集合，不重不漏！

本页要点

只要证明“终点在 S_0、S_0 一步必离开、S_1 总能一步回 S_0”，就彻底证明了 Bouton 定理！

模板代码：标准 Nim 博弈 $O(n)$ 判定

```cpp
int main() {
    ios::sync_with_stdio(false); cin.tie(nullptr); // 关闭同步加速读入
    int n;
    if (!(cin >> n)) return 0;                     // 读入石子堆数 n
    int s = 0;                                     // 异或和初值必须是 0（0^x = x）
    for (int i = 0, x; i < n; ++i) {               // 逐堆读入每堆石子数
        cin >> x;                                  // 读入当前堆石子数 x
        s ^= x;                                    // 累加按位异或到总状态 s
    }
    cout << (s != 0 ? "win" : "lose") << '\n';     // 非 0 先手必胜，为 0 先手必败
    return 0;
}
```

本页要点

初值 s = 0（异或单位元 $x \oplus 0 = x$），循环里 s ^= x，最后判 s != 0 即可！

上机验证与反事实追问

测试类型	输入（堆数 / 各堆）	预期输出	手算核对说明
普通必胜	`4 / 7 12 9 15`	`win`	$7 \oplus 12 \oplus 9 \oplus 15 = 13 \neq 0$
最小边界	`1 / 5`	`win`	只有 1 堆 5 颗，$s = 5 \neq 0$，一把拿光
经典必败	`3 / 1 2 3`	`lose`	$1 \oplus 2 \oplus 3 = 0$，先手无论怎么拿都输

反事实追问

若误把 `s ^= x` 写成 `s += x` 并判 `s % 2 != 0`，输入 `3 / 1 2 3` 会输出什么？
答案：$1+2+3=6$ 也是偶数，看似蒙对；但输入 `2 / 2 4` 时和为偶数而 $2 \oplus 4 = 6 \neq 0$，求和法就会误判成 `lose`！

本页要点

千万不要用“总石子数奇偶性”代替“异或和是否为 0”，反例就是 $(2, 4)$！

Nim 博弈易错对照与适用条件

常见错误场景	错误写法	正确做法与适用提醒
用加法代替代替异或	`s += x` 判断总和奇偶	只能用按位异或 `s ^= x`（WA）
每堆限制了最多拿 k 颗	依然直接做 `a[i]` 的异或	每堆有上限时不是标准 Nim，需先对每堆算 SG 值（$\mathrm{mod}(k+1)$）再异或
“拿光最后一颗算输”	反常 Nim 仍直接判 `s != 0`	全是 1 堆时需看堆数奇偶；存在 ≥ 2 堆时仍与 $s \neq 0$ 一致
输入数据超过 `int`	$a_i \leq 10^{18}$ 仍用 `int s`	若题面石子数达 10^{18}，需改用 `long long`（WA）

本页要点

标准 Nim 要求“每次选一堆拿任意正整数个”；一旦规则加了限制，就要请出第 4 节的 SG 函数！

复杂规则引入：如果棋子在有向图上走呢？

新场景

给定一张有向无环图（DAG），棋盘上有 k 枚棋子。两人轮流选任意一枚棋子沿有向边走一步，无法走动者输。

- 这时候还能把棋子所在编号直接异或吗？绝对不行！节点编号 $1, 2, 3$ 只是名字，不代表剩几步能走。
- 比如节点 5 没有出边，它等价于 0 颗石子的 Nim 堆；节点 4 只连向 5，走一步就死，它等价于 1 颗石子的 Nim 堆！
- 我们真正需要一个“翻译器”：给每个顶点 u 算一个非负整数 $\mathrm{SG}(u)$，告诉我们“位于 u 的这枚棋子等价于几颗石子的 Nim 堆”。

本页要点

只要把每个局面 u 翻译成“等价 Nim 堆大小” $\mathrm{SG}(u)$，多枚棋子就能直接套用 Nim 异或！

为什么需要 SG 定理？算一算联合状态的爆炸速度

- 假设图有 $N = 2000$ 个点，棋盘上放了 $K = 20$ 枚（甚至 2000 枚）棋子。
- 若把“K 枚棋子位置元组 $(u_1, \ldots, u_K)$”当成大状态去倒推，状态数高达 $N^K = 2000^{20}$，必然 **TLE / MLE**！
- 若分别算单枚棋子在各点的 SG(u)：整张图只需 $O(N + M)$ 跑一次记忆化搜索，再用 $O(K)$ 异或合并！

核心思想

Sprague-Grundy（SG）定理把指数级的“联合博弈”拆成线性的“单个独立子游戏”，最后只需异或合并！

本页要点

SG 定理的最大威力是“分而治之”：先用 $O(N + M)$ 算单个图的 SG 值，再用 $O(K)$ 异或合并。

新术语大白话：mex 运算与 SG 函数定义

- **mex（Minimum Excluded）**：给定一个非负整数集合 S，$\mathrm{mex}(S)$ 就是没有在 S 里出现过的最小非负整数（从 0 开始往上数，第一个缺的数就是它）。
- 例如：$\mathrm{mex}(\{0,1,3\}) = 2$（缺 2）；$\mathrm{mex}(\{1,2\}) = 0$（连 0 都缺！）。
- **SG 函数定义**：对于 DAG 中的每个状态 u，它的 SG 值等于所有一步能走到的后继状态 v 的 SG 值集合的 mex：

$$\mathrm{SG}(u) = \mathrm{mex}\{\mathrm{SG}(v) \mid u \to v\}$$

- 特别地，当 u 没有出边（终点）时，后继集合为空集 $\varnothing$，所以 $\mathrm{SG}(u) = \mathrm{mex}(\varnothing) = 0$！

本页要点

mex 的大白话就是“从 0 开始找第一个没出现的数”；终点没有后继，SG 值规定为 0。

mex 手工计算表：与 P/N 态、等价 Nim 堆的对应

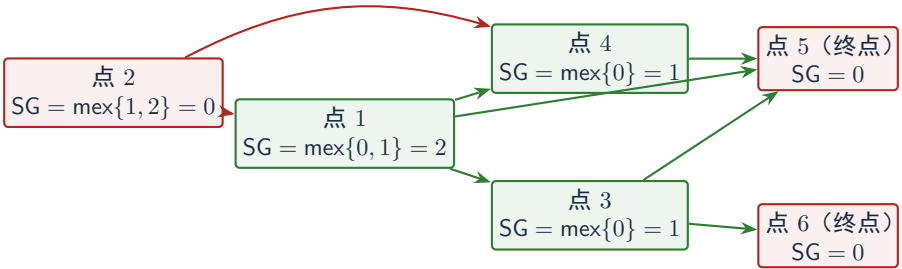
后继 SG 集合 S	mex(S) 结果	局面的 P/N 判定	等价含义说明
$\varnothing$（无出边）	0	**P** 态	无路可走，等价于 0 颗石子的空堆
$\{1,2\}$	0	**P** 态	虽然有后继，但后继全是 N 态（没有 0）
$\{0\}$	1	**N** 态	只能走到 0（P 态），等价于 1 颗石子
$\{0,2\}$	1	**N** 态	缺的最小非负整数是 1（跳过没缺的 0）
$\{0,1,3\}$	2	**N** 态	能走到 0 和 1，最小未出现的是 2
$\{0,1,2\}$	3	**N** 态	能走到 $0,1,2$，恰好等价于 3 颗石子的堆

- 仔细看前两行：$\mathrm{SG}(u)=0 \iff$ 后继里没有 $0 \iff$ 所有后继都是 N 态 $\iff$ u 是 **P** 态！

本页要点

当只有 1 枚棋子时，$\mathrm{SG}(u)=0$ 就是必败态（P），$\mathrm{SG}(u)>0$ 就是必胜态（N）！

TikZ 图解：在 6 点 DAG 上倒推计算 sg[u]



- 倒推顺序：$5, 6$ 无出边 $\to \mathrm{SG} = 0$；接着 $3, 4$ 只能到 $0 \to \mathrm{SG} = 1$；点 1 连向 $3, 4, 5$（SG 为 $1, 1, 0$）$\to \mathrm{SG} = 2$；点 2 连向 $1, 4$（SG 为 $2, 1$）$\to \mathrm{SG} = 0$！

本页要点

求 DAG 上某点 SG 值的过程，本质就是“先算完所有出边终点，再求这些值的 mex”。

为什么多游戏组合等价于 $\mathrm{SG}(u_1) \oplus \cdots \oplus \mathrm{SG}(u_k)$？

- 假设局面 u 的 $\mathrm{SG}(u) = x$，它为什么真的能当成“x 颗石子的 Nim 堆”来异或？
- **1.** 能减到任意更小值：按 mex 定义，$0, 1, \ldots, x-1$ 都在后继里出现过，需要减小到任意 $y < x$ 时真有路可走！
- **2.** 走到更大的值也不怕：若对手从 u 走到 $\mathrm{SG}(v) = z > x$，因 z 的后继里必有 x，你立刻从 z 走回 x 就能抵消！

Sprague-Grundy 定理

多个独立有向图游戏组合的总局面，先手必胜 $\iff \mathrm{SG}(u_1) \oplus \mathrm{SG}(u_2) \oplus \cdots \oplus \mathrm{SG}(u_k) \neq 0$！

本页要点

能走到所有更小 SG 值，且走到更大 SG 值能被一步拉回——这就是 SG 等价 Nim 堆的根基。

记忆化求 sg[u] 的两个生死细节

细节 1：为什么 sg[] 初值必须设 -1，绝对不能设 0？

因为所有必败态（以及终点）的真实 SG 值恰恰就是 0！如果用 0 表示“还没算过”，程序每次遇到 $\mathrm{SG}(v)=0$ 的点都会以为它“没算过”而反复重新递归，在稠密 DAG 上直接超时（**TLE**）！设为 -1 才能严格区分“未计算”与“算出来等于 0”。

细节 2：为什么 seen[] 必须在每层递归局部创建？

假设你把 seen[] 开成全局数组：在算点 u 时，先递归点 v_1，回来标记了 seen[0]=1；接着递归点 v_2，如果 v_2 内部又去清空或修改了同一个全局 seen[]，刚给 u 记好的标记就会被下层递归踩坏！每层局部开 vector<char> seen(g[u].size() + 2, 0)，空间只占出度大小，安全且总空间可控！

本页要点

记住这两条铁律：sg 初值必设 -1，seen 必须每层局部开！

模板代码：DAG 记忆化搜索求 sg[u]

```
vector<int> sg(n + 1, -1);                          // 必须初始化为 -1，区分未算与 SG=0
auto get_sg = [&](auto &&self, int u) -> int {      // 记忆化搜索计算顶点 u 的 SG 值
    if (sg[u] != -1) return sg[u];                  // 已计算过则直接返回，保证线性复杂度
    vector<char> seen(g[u].size() + 2, 0);          // 局部标记数组，绝不与下层递归共用
    for (int v : g[u]) {                            // 遍历 u 的所有有向后继 v
        int val = self(self, v);                    // 递归求出后继 v 的 SG 值
        if (val < (int)seen.size()) seen[val] = 1; // 越界保护：超过出度的 mex 不影
    响结果
    }
    int mex = 0;                                    // 从 0 开始寻找最小未出现非负整数
    while (seen[mex]) ++mex;                        // 遇到出现过的就跳过
    return sg[u] = mex;                             // 存入记忆化数组并返回
};
```

本页要点

局部 seen 长度为“出度 + 2”（抽屉原理保证 mex $\leq$ 出度），总空间 $O(N+M)$ 且绝不踩踏！

上机验证、反事实追问与易错对照

测试/错误场景	错误写法或输入	预期结果 / 评测后果
单终点测试	1 点 0 边 1 枚棋子在点 1	$\mathrm{SG}(1)=0$，输出 `lose`
同点双棋子	2 点 1 边 $(1\to 2)$，2 枚都在点 1	$\mathrm{SG}(1)=1$，但 $1\oplus 1=0$，输出 `lose`
`sg` 初值设为 **0**	`vector<int> sg(n+1, 0)`	$\mathrm{SG}=0$ 的点反复重算，大图 **TLE**
全局共用 `seen`	全局 `bool seen[]` 函数开头清空	递归子调用覆盖上层标记，小图对大图 **WA**

反事实追问

两枚棋子都放在 $\mathrm{SG}(u)=1$ 的同一点 u 上，先手会赢吗？
答案：不会！它们等价于两堆各 1 颗石子（$1\oplus 1=0$），后手只需模仿先手移动另一枚棋子，先手必败！

本页要点

看到多枚棋子可重合在同一点时，务必每枚各异或一次，绝对不能去重！

【例 5.1】[基础] 取石子游戏 1 | 先读懂要求

大白话题意（一本通 1663）

2 名绝顶聪明的玩家面对 N 颗石子轮流拿，每次最少拿 1 颗、最多拿 K 颗；拿到最后一颗石子的一方获胜。

- 输入：一行两个整数 N, K（$1 \le N \le 10^5$，$1 \le K \le N$）。
- 处理：判断在双方最优策略下，是先手必胜还是后手必胜。
- 输出：一行一个整数，先手胜输出 1，后手胜输出 2。
- 原题样例：输入 23 3，输出 1。

本页要点

先看清输出格式要求：这道题输出数字 1 或 2，不是 win/lose！

【例 5.1】取石子游戏 1 | 怎么想到的与分组图解

- 为什么不用循环倒推？虽然 $N \le 10^5$，但若 K 也达 10^5，直接枚举 $1 \sim K$ 会变成 $O(NK) = 10^{10}$ 超时。
- 一眼认出这是标准巴什博弈：把 23 颗石子按每组 $K+1=4$ 颗拆分！



- $23 = 5 \times 4 + 3$：先手第一轮拿掉 3 颗，后面 5 组每组都由“后手拿 x、先手补 $4-x$”成对清掉！

本页要点

只要 N 不是 $K+1$ 的倍数，先手第一手把余数拿走，就锁死了整局胜势。

【例 5.1】取石子游戏 1 | 样例 $N = 23, K = 3$ 手推跟踪表

阶段	行动方	操作前剩余	拿走颗数	操作后剩余（余数）
第 1 步	先手	23	3（$= 23 \bmod 4$）	20（$\bmod 4 = 0$）
第 1 组	后手 $\to$ 先手	20	后手 $x \to$ 先手 $4 - x$	16（$\bmod 4 = 0$）
第 2 组	后手 $\to$ 先手	16	后手 $y \to$ 先手 $4 - y$	12（$\bmod 4 = 0$）
第 3-4 组	后手 $\to$ 先手	12	连续两轮凑满 4	4（$\bmod 4 = 0$）
第 5 组	后手 $\to$ 先手	4	后手 $z \to$ 先手 $4 - z$	0（先手拿走最后一颗！）

- 无论后手每组选 $1, 2, 3$ 中的哪一个，先手总能补齐 4，所以第 5 组结束时先手拿光并输出 1。

本页要点

手推时把每组“后手 $x +$ 先手 $(K + 1 - x)$”合在一起看，5 步就直达终点 0！

【例 5.1】取石子游戏 1 ｜核心步骤与模板改动

步骤	按顺序计算	纸上得到	核对问题
1	读入 n 和 k	$n=23, k=3$	数据在 10^5 内，`int` 足够
2	计算 $n \bmod (k+1)$	$23 \bmod 4 = 3$	括号是否写全？不能写成 `n%k+1`
3	判断余数是否非 0	$3 \neq 0$ 为真	非 0 输出 `1`，为 0 输出 `2`

相对第 2 节巴什模板改了哪里

1. 把输出字符串从 `"First Win"`/`"Second Win"` 改为题面要求的整数 `1`/`2`；
2. 因本题 $N \leq 10^5$，使用 `int` 即可（若数据升至 10^9 以上则改用 `long long`）。

本页要点

写取余表达式时，括号（`k + 1`）绝对不能省，否则会算成（`n % k`）`+ 1`！

【例 5.1】取石子游戏 1 ｜核心代码（code/bash_game.cpp）

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

// 一本通 1663 取石子游戏 1（巴什博弈）：n%(k+1)!=0 先手胜(1)，否则后手胜(2)
int main() {
    ios::sync_with_stdio(false); cin.tie(nullptr); // 加速读入输出
    int n, k;                                      // 定义石子总数 n 与上限 k
    if (!(cin >> n >> k)) return 0;                // 读入 n 和 k
    if (n % (k + 1) != 0) cout << 1 << '\n';       // 余数非 0，先手拿余数必胜
    else cout << 2 << '\n';                        // 恰好整除，先手必败输出 2
    return 0;
}
```

本页要点

代码与 code/bash_game.cpp 一致；注意 n % (k + 1) 括号及输出 1/2 的对应关系。

【例 5.1】取石子游戏 1 | 易错与 3 组自测边界

自测类型	输入 (N, K)	预期输出	检验目的
最小边界	`1 1`	1	$1 \bmod 2 = 1 \neq 0$，先手拿 1 颗赢
整除必败	`4 3`	2	$4 \bmod 4 = 0$，检验 2 分支是否正确
满拿边界	`100000 100000`	1	$K = N$ 时 $N \bmod (N+1) = N \neq 0$，一把拿完

高频错误提醒

把 `n % (k + 1)` 漏括号写成 `n % k + 1`：不仅优先级错，遇到任何输入结果恒大于 0，导致永远输出 1（全片 **WA**）！

本页要点

交题前一定测一组 $N = K + 1$（如 `4 3`），确认程序真的能输出 2！

【例 5.2】[基础] 取石子游戏 2 | 先读懂要求

大白话题意（一本通 1664）

有 N 堆石子，每堆有 $X_1, \ldots, X_N$ 颗。两人轮流选某一堆拿走至少 1 颗，轮到某人时已无石子可拿则判负。

- 输入：第一行一个整数 N（$N \le 5 \times 10^4$）；第二行 N 个整数 X_i（$1 \le X_i \le 10^5$）。
- 处理：计算所有堆石子数的按位异或和，判断是否非 0。
- 输出：先手胜输出 `win`，后手胜输出 `lose`（全小写）。
- 原题样例：输入 `4` 与 `7 12 9 15`，输出 `win`。

本页要点

看到“多堆、每轮只选一堆拿任意正整数个”，直接对应标准 Nim 博弈模型！

【例 5.2】取石子游戏 2 | 怎么想到的与二进制位对齐图

- 把原题样例 7，12，9，15 拆成 4 位二进制，看看各位上 1 的个数是不是偶数：

$X_1 = 7$:	0	1	1	1
$X_2 = 12$:	1	1	0	0
$X_3 = 9$:	1	0	0	1
$X_4 = 15$:	1	1	1	1
异或和 $s = 13$:	1	1	0	1

第 3、2、0 位上 1 的个数都是奇数！
选 $X_4 = 15$ 改成 $15 \oplus 13 = 2$（拿走 13 颗），
新局面 $(7, 12, 9, 2)$ 异或和立刻变为 0！

本页要点

先手在样例 $(7, 12, 9, 15)$ 中只需从 15 里拿走 13 颗变成 2，就能把平衡态 0 甩给后手！

【例 5.2】取石子游戏 2 | 样例 7 12 9 15 手推跟踪表

循环轮次 i	读入石子数 x	计算前 s	计算后 $s = s \oplus x$	二进制变化轨迹
初值	—	—	0	`0000`
$i = 0$	7	0	7	`0000 ^ 0111 = 0111`
$i = 1$	12	7	11	`0111 ^ 1100 = 1011`
$i = 2$	9	11	2	`1011 ^ 1001 = 0010`
$i = 3$	15	2	13	`0010 ^ 1111 = 1101`（非 0!）

- 循环结束条件 $i = 4$ 为假退出，最终 $s = 13 \neq 0$，因此输出 `win`，与原题样例输出完全吻合！

本页要点

用 4 步异或跟踪表，能边读入边累加异或，连数组都不用开！

【例 5.2】取石子游戏 2 | 核心步骤与模板改动

步骤	按顺序计算	纸上得到	核对问题
1	读入堆数 n，设 $s=0$	$n=4, s=0$	初值是否为 0？
2	循环 n 次读 x 并执行 $s\oplus=x$	s 依次为 $7\to11\to2\to13$	运算符是否为 `^=` 而非 `+=`？
3	判断 $s\neq0$ 并输出	$13\neq0\to$ `win`	大小写是否全小写？

相对第 3 节 Nim 模板改了哪里

1. 本题即标准 Nim 博弈的直接应用，无需开数组保存 X_i，空间复杂度仅 $O(1)$；
2. 注意 $N\le5\times10^4$，使用 `ios::sync_with_stdio(false); cin.tie(nullptr);` 加快读入。

本页要点

边读边异或只需 $O(1)$ 空间与 $O(N)$ 时间，是最干净的 Nim 实现。

【例 5.2】取石子游戏 2 | 核心代码（code/nim_game.cpp）

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

// 一本通 1664 取石子游戏 2（Nim 博弈）：异或和 != 0 先手必胜(win)，== 0 先手必败(
    lose)
int main() {
    ios::sync_with_stdio(false); cin.tie(nullptr); // 加速大量数据读入
    int n; if (!(cin >> n)) return 0;              // 读入石子堆数 n
    int s = 0;                                     // 异或和初值设为 0
    for (int i = 0, x; i < n; ++i) {               // 依次读入 n 堆石子
        cin >> x; s ^= x;                          // 将当前堆石子数异或进总和 s
    }
    cout << (s != 0 ? "win" : "lose") << '\n';     // 非 0 输出 win，否则 lose
    return 0;
}
```

完整代码见 code/nim_game.cpp，已通过 g++ -O2 -std=c++17 编译；注意 "win"/"lose" 全小写。

【例 5.2】取石子游戏 2 | 易错与 3 组自测边界

自测类型	输入数据	预期输出	检验目的
最小必胜	`1 / 1`	`win`	只有 1 堆 1 颗，$s=1\neq 0$，直接拿完
对称必败	`2 / 5 5`	`lose`	相等两堆 $5\oplus 5=0$，检验 `lose` 分支
原题样例	`4 / 7 12 9 15`	`win`	核对多堆非对称必胜情况

高频错误提醒

1. 把异或 `s ^= x` 误写成加法 `s += x`，或误以为只要有奇数堆就必胜；
2. 输出写成 `"Win"`/`"WIN"` 导致格式不符（**WA**）。

本页要点

遇到多堆且每堆可任取的问题，永远先算异或和，绝不凭“奇偶直觉”猜胜负。

【例 5.3】[提高] 移棋子游戏 | 先读懂要求

大白话题意（一本通 1665）

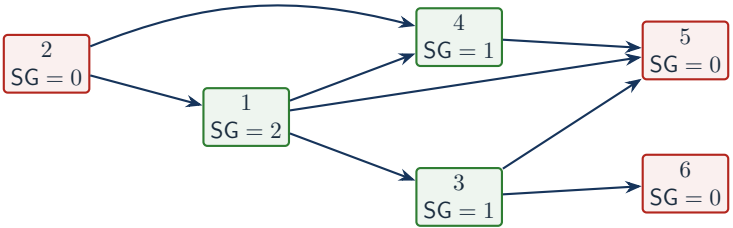
给定一个 N 个点、M 条边的有向无环图（DAG），棋盘上有 K 枚棋子（可放在同一顶点）。两人轮流选一枚棋子沿有向边走一步，无法移动者输。

- 输入：第一行 N, M, K（$N \le 2000, M \le 6000, 1 \le K \le N$）；接下来 M 行每行一条有向边 $X \to Y$；最后一行 K 个整数表示各棋子初始顶点。
- 处理：用记忆化搜索求每个顶点的 SG 值，再将 K 枚棋子所在顶点的 SG 值全部异或。
- 输出：先手胜输出 `win`，否则输出 `lose`。

本页要点

多枚棋子互不干扰，每枚棋子就是一个独立的有向图子游戏！

【例 5.3】移棋子游戏 | 原题样例 DAG 图解（$N = 6, M = 8$）



- 原题样例给出的 8 条边是：2->1，2->4，1->4，1->5，4->5，1->3，3->5，3->6，棋子在 1，2，4，6。
- 站在点 1 看，它连向 $3, 4, 5$；必须先算出点 $3, 4, 5$ 的 SG 值，才能求点 1 的 mex！

本页要点

有向图求 SG 就是沿着有向边递归下去，回溯时对后继集合求 mex。

【例 5.3】移棋子游戏 | 样例手推跟踪表

顶点 u	出边后继 v	后继 SG 值集合	mex 结果	说明
5	无	$\varnothing$	$\mathrm{SG}[5]=0$	终点无路可走
6	无	$\varnothing$	$\mathrm{SG}[6]=0$	终点无路可走
4	$\{5\}$	$\{0\}$	$\mathrm{SG}[4]=1$	只能走到 0，缺 1
3	$\{5,6\}$	$\{0,0\}$	$\mathrm{SG}[3]=1$	两个后继都是 0，仍缺 1
1	$\{4,5,3\}$	$\{1,0,1\}$	$\mathrm{SG}[1]=2$	有 0 和 1，最小未出现是 2
2	$\{1,4\}$	$\{2,1\}$	$\mathrm{SG}[2]=0$	后继缺 0，因此是 P 态！

- 最后合并 4 枚棋子（位于 1，2，4，6）：
  $\mathrm{SG}[1]\oplus\mathrm{SG}[2]\oplus\mathrm{SG}[4]\oplus\mathrm{SG}[6]=2\oplus0\oplus1\oplus0=3\neq0 \implies$ 输出 `win`！

本页要点

哪怕有 2 条边算出同样的后继 SG 值（如点 3 连向两个 0），也不影响 mex 结果！

【例 5.3】移棋子游戏｜核心步骤与模板改动

步骤	按顺序计算	纸上得到	核对问题
1	建邻接表 g[u] 存 M 条边	6 点 8 边邻接表	有向边只加 g[u].push_back(v)
2	设 sg 全 -1，写 get_sg	$\mathrm{SG} = [2, 0, 1, 1, 0, 0]$	是否局部开 seen 且设初值 -1？
3	读 K 枚棋子并异或 get_sg	$2 \oplus 0 \oplus 1 \oplus 0 = 3$	同一点多枚棋子是否都参与异或？

相对第 4 节 SG 模板改了哪里

把单点查询扩展为循环读入 K 枚棋子坐标，用 `xr ^= get_sg(get_sg, u)` 累乘异或总和，最后按 `xr != 0` 输出 `win/lose`。

本页要点

用邻接表存图时，有向边不要误建成无向双向边，否则会出现环导致无限递归！

【例 5.3】移棋子游戏｜核心代码（1/2：建图与 get_sg）

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

// 一本通 1665 移棋子游戏：有向图上的 SG 函数 + 多枚棋子 SG 值异或总和
int main() {
    ios::sync_with_stdio(false); cin.tie(nullptr); // 加速读入
    int n, m, k; if (!(cin >> n >> m >> k)) return 0; // 读点数、边数、棋子数
    vector<vector<int>> g(n + 1);                  // 邻接表存有向无环图
    for (int i = 0, u, v; i < m; ++i) {            // 读入 m 条有向边
        cin >> u >> v; g[u].push_back(v);          // 添加单向边 u -> v
    }
    vector<int> sg(n + 1, -1);                     // 初值必须为 -1，不能是 0
```

本页要点

先建单向邻接表并设 sg 初值 -1；下一页展示记忆化求 mex 与多棋子异或。

【例 5.3】移棋子游戏｜核心代码（2/2：记忆化与异或）

```
    auto get_sg = [&](auto &&self, int u) -> int { // 递归求 u 的 SG 值
        if (sg[u] != -1) return sg[u];             // 算过直接返回
        vector<char> seen(g[u].size() + 2, 0);     // 每层局部开 seen 防踩踏
        for (int v : g[u]) {                       // 遍历所有后继节点
            int val = self(self, v);               // 递归求后继 SG 值
            if (val < (int)seen.size()) seen[val] = 1; // 越界保护标记
        }
        int mex = 0; while (seen[mex]) ++mex;      // 找最小未出现非负整数
        return sg[u] = mex;                        // 记忆化并返回
    };
    int xr = 0;                                    // 多枚棋子 SG 异或总和
    while (k--) { int u; cin >> u; xr ^= get_sg(get_sg, u); } // 独立子游戏异或
    cout << (xr != 0 ? "win" : "lose") << '\n';    // 非 0 先手胜
    return 0;
}
```

完整代码见 `code/sg_chess.cpp`（1/2 与 2/2 拼接），已通过 `g++ -O2 -std=c++17` 编译与实测。

【例 5.3】移棋子游戏｜易错与 3 组自测边界

自测类型	输入数据	预期输出	检验目的
单点无边	`1 0 1 / 1`	`lose`	终点 SG $=0$，直接判负
同点双棋	`2 1 2 / 1 2 / 1 1`	`lose`	两枚同在必胜点 1，$1 \oplus 1 = 0$ 必败
原题样例	`6 8 4`（原题 8 边 + `1 2 4 6`）	`win`	验证复杂有向图与多棋子异或

高频错误提醒

1. 把 K 枚棋子所在点去重（题目允许多个棋子在同一点，去重会漏掉偶数次抵消!）；
2. `seen[val] = 1` 前忘记判断 `if (val < (int)seen.size())`，若某后继出度极大导致 `val >= seen.size()` 会触发越界（**RE**）。

本页要点

同一点放两枚棋子时，它们互相独立且等价于两堆相等石子，异或后正好抵消为 0!

博弈论与 SG 函数常见错误对照表

题号 / 场景	常见错误写法	正确做法与原因
1663 取石子 1	写成 `n % k != 0` 或漏括号	每轮凑对和是 $k+1$，必须写 `n % (k + 1) != 0`
1664 取石子 2	用总和奇偶 `s += x` 代替异或	必须用按位异或 `s ^= x`，$(2,4)$ 即可证伪奇偶法
1665 移棋子（记忆化）	`vector<int> sg(n+1, 0)`	终点 SG 本来就是 0，初值必须设 `-1` 防重复递归
1665 移棋子（mex）	共用全局 `seen[]` 数组	子递归会踩坏上层标记，必须每层局部开 `seen`
多棋子同坐标	对初始棋子位置去重	棋子互相独立，同一点放 2 枚应异或抵消为 0

本页要点

博弈代码短但细节致命：`k+1` 括号、`^=` 异或、`sg` 初值 `-1`、局部 `seen` 四处最常出事。

按评测结果（CE / WA / PE / RE / TLE）快速定位

评测结果	本章常见诱因	定位与排查顺序
CE	匿名递归 `auto &&self` 语法	确认开启 C++14/17（`-std=c++17`），或改写成普通全局函数
WA	全局 `seen` 踩踏 / 输出大小写	测 `1 2 3`、`2 4` 与同点双棋子；核对 `1/2` 与 `win/lose`
PE	末尾多余空格或漏换行	每次只输出一个结果并带 `'\n'`
RE	`seen[val]` 越界或无向建边爆栈	标记前加 `if (val < (int)seen.size())`；确认只加单向边
TLE	`sg[]` 初值写成 0 或暴力搜状态	检查记忆化初值是否为 `-1`，且是否漏了 `if (sg[u]!=-1)`

本页要点

SG 莫名 TLE 首先查 sg 初值是不是 0；小图对大图 WA 首先查 seen 是不是全局共用！

提交前检查清单（六必查）

1. 巴什博弈取余时，`n % (k + 1)` 的括号是否完整（没有漏成 `n % k + 1`）？
2. Nim 博弈累加变量 `s` 初值是否为 `0`，且用的是 `^=` 而不是 `+=`？
3. 记忆化搜索 `sg[]` 数组的未访问初值是否设为了 `-1`（而非 `0`）？
4. 求 `mex` 时的 `seen[]` 是否每层局部创建，且标记前有 `val < (int)seen.size()` 越界保护？
5. 多枚棋子在同一点重合时，是否每枚棋子都各自异或了一次（没有错误去重）？
6. 输出格式（`1/2` 还是全小写 `win/lose`）是否与题目样例完全一致？

最后一步自测

提交前手敲两组必败样例验证：`1 2 3`（输出 `lose`）与“两点一条边、两枚棋子同在起点”（$1 \oplus 1 = 0$，输出 `lose`）。

本章全部练习与完成标准

序号	题目名称（一本通 / 题库）	难度标签	核心考察点与思路提要
1	取石子游戏 1（一本通 1663）	[基础]	巴什博弈，$n \bmod (k+1) \neq 0$ 先手胜输出 `1`
2	取石子游戏 2（一本通 1664）	[基础]	标准 Nim 博弈，全体异或和 $s \neq 0$ 输出 `win`
3	移棋子游戏（一本通 1665）	[提高]	DAG 记忆化求 `sg[u]` + 多棋子 `sg` 异或
4	S-Nim（提高篇数学基础）	[提高]	给定允许取子集合 S，预处理各堆 `sg[x]` 后异或
5	取石子游戏（一本通 1666）	[提高]	把规则变化拆成独立子局面并套用 `mex` 规律

本章完成标准

1. 能用自己的话向队友讲清“为什么异或和为 0 是必败态”；
2. 能不看模板、零失误默写出带局部 `seen` 与初值 `-1` 的 `get_sg` 函数。

课后安排与赛场衔接建议

- 第 **1** 天（巩固）：独立 AC 一本通 1663、1664、1665 三道配套真题，并用前文的边界自测数据检验。
- 第 **2–3** 天（拓展）：完成 S-Nim 与 取石子游戏（1666），尝试“先写暴力 SG 打表找规律，再推 $O(1)$ 公式”。
- 赛场实战：这是解决复杂博弈最实用的套路——比赛时先写个小范围 `get_sg` 打表输出前 30 项，往往能一眼看出周期或异或规律！
- 周赛习惯：从本周起，坚持每周打一场 AtCoder Beginner Contest（ABC）或 Codeforces Div.3/Div.2，赛后把做不出的第 1 道题补透！

本页要点

“暴力打表找规律 + 异或/SG 定理证明”，是正式比赛中最稳的博弈解题双保险。

全系列 8 篇总览回顾：从零基础到算法竞赛主力

篇目	核心模块	你已经掌握的硬核武器
01	C++ 语言与 STL	语法、循环、函数、结构体，以及 `vector/set/map/priority_queue`
02	基础算法与思维	复杂度估算、前缀和差分、双指针、排序、二分、贪心与分治
03	搜索专题	DFS 回溯剪枝、BFS 最短路、状态去重与双向/启发式搜索
04	数据结构	栈队、单调结构、并查集、ST 表、树状数组、线段树、LCA、平衡树
05	动态规划	线性、背包、区间、树形、状压与单调队列/斜率优化
06	图论算法	最短路、最小生成树、拓扑序、强连通分量与二分图匹配
07	字符串算法	字符串哈希、KMP、字典树 Trie 与 AC 自动机
08	数学基础	质数筛法、同余逆元、矩阵快速幂、容斥期望与博弈论 **SG**

本页要点

八大专题环环相扣：语言是工具，基础与搜索打底，数据结构与 DP 图论成体系，字符串与数学封顶！

全系列收官寄语与最后一句话

- 恭喜你走完了整套《HGNU-ACM 算法竞赛零基础到进阶》全部八篇课程！
- 真正拉开差距的从来不是“看过多少课件”，而是每道模板自己敲过多少遍、每道错题有没有在草稿纸上追踪过变量。
- 遇到难题卡住时，回到最初的笨办法：先手算 $n = 1, 2, 3, 4$ 的小例子，画出状态图，写出跟踪表，真相自然会浮出来。

本章（全系列）一句话

组合博弈把胜负拆解成清晰的 P/N 状态与异或平衡；愿你在未来的每一场算法竞赛中，都能像找准制胜步一样沉着果断——全系列完结，赛场再见！