单调队列优化 DP

分离决策变量 + 滑动窗口淘汰，将转移从 $O(n^2)$ 降至 $O(n)$

HGNU-ACM 实验室算法组

承接第 4 章《状压与数位 DP》丨配套原理拆解、多重背包优化与 4 道真题精讲





本章目录

滑动窗口转移与决策变量分离

队头过期与队尾单调性淘汰

多重背包按余数分组的单调队列优化

前三节零基础拆解：从“滑动窗口为什么 TLE”到“队头过期与队尾淘汰”，再到“多重背包余数分组”。

本章目录（续）

单调队列优化 DP 真题精讲

常见错误、检查清单与练习安排

后两节精讲《最大连续和》《烽火传递》《修剪草坪》《绿色通道》并总结常见错误与检查清单。

生活引入：连续 m 天的最低气温

承接第 4 章

第 4 章我们把整行棋盘或十进制数位压成了状态；本章回到一维序列 DP，解决“每次转移都要扫一段区间求最值”的超时问题。

- 想象气象站记录了连续 n 天的气温，每天你都要看“包括今天在内最近 m 天的最低气温”，再结合今天的湿度算出一个指数。
- 笨办法：第 i 天回头翻前 m 天的记录找最小；第 $i+1$ 天又回头翻前 m 天——其中有 $m-1$ 天昨天刚翻过！
- 单调队列干的事就是在窗口向右滑的时候，把有用的历史最值留在队伍里，新的一天只需 $O(1)$ 看一眼队头！

本页要点

单调队列优化的本质，就是用双端队列消除相邻两次窗口查询之间的重复扫描。

暴力转移为什么会 TLE？

- 考虑经典的定宽窗口转移方程：

$$dp[i] = g[i] + \min_{i-m \le j \le i-1} f[j]$$

- 当数据范围达到 $n = 2 \times 10^5, m = 10^5$ 时：
  - 外层循环 i 从 1 到 n 共 2×10^5 轮；
  - 内层暴力 `for (int j = max(0, i-m); j < i; ++j)` 每轮扫最多 m 次；
  - 总操作次数约 $n \times m \approx 2 \times 10^{10}$ 次，远超 1 秒上限（约 10^8 次），必 **TLE**！
- 用单调队列维护窗口 $[i-m, i-1]$ 内的最小 $f[j]$ 后，总操作次数降为均摊 $O(n)$（约 4×10^5 次），5 毫秒内稳过！

本页要点

只要内层循环是在一个右移的区间 $[i-m, i-1]$ 里找最值，就能从 $O(nm)$ 降成 $O(n)$。

新术语大白话：决策变量分离 f[j] + g[i]

核心定义

- 决策点 j：算第 i 个状态时，用来转移的前驱下标 j（例如 $j \in [i-m, i-1]$）。
- 决策变量分离：把转移方程里的表达式拆成只含 j 的部分 $f[j]$ 和只含 i 的部分 $g[i]$，形式为：

$$dp[i] = \min_{i-m \le j \le i-1} \{f[j]\} + g[i]$$

- 为什么要“分离”？因为 $g[i]$ 对于同一个 i 是固定常数，可以提到 min 外面！
- 括号里只剩 $f[j]$，不同 i 对两个候选 $j_1 < j_2$ 谁更优的评价永远不变，才能放心淘汰劣质决策！

本页要点

口诀：只跟 j 有关的留进队列里比较，只跟 i 有关的提到最值外面加！

图解：能不能用单调队列的唯一分界线



本页要点

看到窗口最值先看有没有 i 和 j 相乘的项：没有就上本章单调队列，有就留给第 6 章斜率优化！

手算跟踪表：暴力扫描的重复劳动（$m = 3$）

小数据设定

设 $f = [0, 3, 1, 4, 2, 5]$，对每个 $i = 1 \ldots 5$ 求 $\min_{i-3 \le j \le i-1} f[j]$。

当前 i	合法窗口 $[i-3, i-1]$	暴力要看哪些 $f[j]$	重复比了谁？
1	$[0, 0]$	$f[0] = 0$	首次计算，最小 0
2	$[0, 1]$	$f[0] = 0, f[1] = 3$	又看了一遍 $f[0]$
3	$[0, 2]$	$f[0] = 0, f[1] = 3, f[2] = 1$	又看了 $f[0], f[1]$
4	$[1, 3]$	$f[1] = 3, f[2] = 1, f[3] = 4$	$f[1] = 3$ 比 $f[2] = 1$ 老又大，还在看！
5	$[2, 4]$	$f[2] = 1, f[3] = 4, f[4] = 2$	最小仍是 $f[2] = 1$
6（止）	$i > n = 5$ 结束	退出循环	暴力总共扫描 15 次

本页要点

从第 4 行可以看出，$f[1] = 3$ 在 $f[2] = 1$ 出现后就已经彻底没用了，但暴力每一轮还在傻傻看它。

正确性论证：为什么可以把 $g[i]$ 提到外面

阶段	等价变形依据：$\min_{j\in W(i)}(f[j]+g[i]) = g[i]+\min_{j\in W(i)} f[j]$
初始化	在确定 i 之前，把所有只依赖于前驱下标 j 的项归并成新数组 $f[j]$。
保持	固定当前 i 时，$g[i]$ 对窗口内所有 j 都是同一个常数，加常数不改变任何两个 $f[j_1], f[j_2]$ 的大小顺序。
终止	求出 $\min f[j]$ 后加上 $g[i]$，得到精确的 $dp[i]$，没有任何精度或最优解损失。

本页要点

正是因为加同一常数 $g[i]$ 不改变 $f[j_1]$ 与 $f[j_2]$ 的大小序，队列里只存下标 j 就够了！

模板代码（1/2）：分离变量与朴素对照

```
// 朴素 O(nm) 写法：帮助看清“待优化的窗口”在哪里
vector<long long> dp(n + 1, 0), f(n + 1, 0), g(n + 1, 0);
for (int i = 1; i <= n; ++i) {
    long long best = LLONG_MAX;
    for (int j = max(0, i - m); j <= i - 1; ++j) {
        best = min(best, f[j]);           // 仅依赖 j 的部分参与窗口最值
    }
    dp[i] = best + g[i];                  // 仅依赖 i 的部分在窗口外加上
    f[i] = dp[i];                         // 根据题意更新 f[i]（如前缀和或 dp[i]）
}
```

本页要点

写单调队列前，先在草稿纸上写出这个两层循环版，确认内层只比 `f[j]` 且范围是 $[i-m, i-1]$。

模板代码（2/2）：单调队列替换内层循环

```
deque<int> q{0};                                  // 放入起点 0，对应 f[0]
for (int i = 1; i <= n; ++i) {
    while (!q.empty() && q.front() < i - m) q.pop_front(); // 1. 踢过期队头
    dp[i] = f[q.front()] + g[i];                  // 2. O(1) 取队头最值并加 g[i]
    f[i] = dp[i];                                 // 3. 算出新决策 f[i]
    while (!q.empty() && f[q.back()] >= f[i]) q.pop_back(); // 4. 踢劣质队尾
    q.push_back(i);                               // 5. 新决策 i 入队
}
```

本页要点

用 4 行队列操作替换了整个内层 `for j`，复杂度从 $O(nm)$ 直接掉到 $O(n)$！

上机验证与反事实追问

3 组标准验证（以 $dp[i] = g[i] + \min_{i-m \le j < i} dp[j]$，$dp[0] = 0$ 为例）

1. 最小边界：$n = 1, m = 1, g = [0, 5] \to$ 从 $dp[0] = 0$ 转移，预期输出 5。
2. 手算对照：$n = 5, m = 2, g = [0, 1, 3, 2, 4, 1] \to$ 对应选 $1, 3, 5$ 号点，预期输出 4。
3. 全退化边界：$m = 1$ 时窗口宽度恒为 $1 \to$ 每步只能从 $i - 1$ 转移，输出全部元素之和。

反事实追问

如果转移方程是 $dp[i] = \min_{i-m \le j < i}\{dp[j] + i \cdot b[j]\}$，强行用单调队列按 $dp[j]$ 排序会怎样？

$\Rightarrow$ 对 $i = 10$ 来说 $dp[1] + 10b[1]$ 可能更大而被踢掉，但到 $i = 100$ 时若 $b[1]$ 很小，它反而反超成为最优！单调队列把未来的最优解提前踢掉，导致 **WA**。

本页要点

只要方程里同时有 i 和 j 相乘，就不能只按 $f[j]$ 淘汰队尾。

变量分离易错对照：WA / TLE / RE 定位

错误类型	常见错误写法	正确做法与原因
WA（忘放 0）	`deque<int> q;` 空队列出发	找不到 $j=0$ 的初始跳板，或取队头时越界
WA（混入 i）	队列里直接存 `dp[j] + i*b[j]`	存在交叉乘积项不能用普通单调队列，应换斜率优化
WA（加错时机）	把 `g[i]` 算进 `f[i]` 前就入队	应先由队头算出 `dp[i]` 和 `f[i]`，再拿 `f[i]` 进队尾比较
TLE（未降维）	只用了 `deque` 但没弹出劣质队尾	队列退化成存了全部 m 个元素，扫描仍然超时

本页要点

记住：队列里存的永远是“下标 j”，比较的是只跟 j 有关的评估值 `f[j]`！

两种淘汰：为什么队头和队尾都要踢人？

- 单调队列之所以能保持“队头永远是当前窗口最优”，靠的是一前一后两种淘汰：

1. 队头过老淘汰：`while (!q.empty() && q.front() < i - m) q.pop_front();`
   - 队头元素的下标已经小于窗口左边界 $i-m$，说明它“滑出窗口”了，再优也不能用！
2. 队尾单调性淘汰：`while (!q.empty() && f[q.back()] >= f[i]) q.pop_back();`
   - 求最小值时，如果队尾那个旧决策 $q.\text{back}()$ 的值 $\geq$ 新决策 $f[i]$：
     新决策 i 既比你更年轻（在窗口里活得更久），又比你更便宜（值更小或相等）！
   - 旧决策在未来任何窗口里都不可能赢过 i，直接永久踢出队列！

本页要点

一句大白话记住队尾淘汰：“如果一个选手比你年轻，还比你强，那你就可以直接退役了！”

图解：滑窗队列的四步演化过程（$m = 3$）



本页要点

队头按“下标过期”淘汰，队尾按“值不如新人”淘汰，队列里永远只留“又近又好”的接班人。

手算跟踪表：$f=[0,3,1,4,2,5], m=3$ 队列完整演化

轮次 i	踢过期后队头	本轮取最小 f[front]	踢劣质队尾	本轮结束队列（下标: 值）
初态	—	—	—	[0:0]
1	0 >= -2 不踢	$f[0]=0$	$0<3$ 不踢	[0:0, 1:3]
2	0 >= -1 不踢	$f[0]=0$	踢 1:3（$3\geq 1$）	[0:0, 2:1]
3	0 >= 0 不踢	$f[0]=0$	$1<4$ 不踢	[0:0, 2:1, 3:4]
4	踢 0:0（$0<1$）	$f[2]=1$	踢 3:4（$4\geq 2$）	[2:1, 4:2]
5（终止）	2 >= 2 不踢	$f[2]=1$	$2<5$ 不踢	[2:1, 4:2, 5:5]

本页要点

注意第 4 轮：先踢掉过期的 0:0 后，刚好由早就排在第二位的 2:1 顺理成章接班成新队头！

均摊 $O(n)$ 证明与单调性不变量

阶段	单调性不变量：队列下标严格递增，且对应 f 值严格递增（求最小时）
初始化	队列仅含 `[0]`，单元素天然严格递增。
保持	队头出队只删首部，不破坏单调性；队尾把所有 $\geq f[i]$ 的弹出后再放入 i，因此新队尾必严格大于前一个元素。
终止	每次循环结束时队列仍保持递增，故队头必为当前窗口内最小决策。

为什么总时间是 $O(n)$ 而不是 $O(n^2)$？

虽然代码里套了两个 `while` 循环，但每个下标 $i \in \{0, 1, \ldots, n\}$ 最多入队 1 次、最多出队 1 次（从队头或队尾出）。总弹出次数不超过 $n+1$，均摊每步 $O(1)$，总时间复杂度严格 $O(n)$！

本页要点

看复杂度不能只数 `while` 层数，要看“每个元素一生进出队列各几次”。

铁律：先查队头再入队，还是先入队再查队头？

题目类型	正确步骤顺序	搞反了会怎样？
DP 转移 / 非空区间（要求 $j < i$，不能自己转自己）	1. 踢过期队头 2. 查队头算 `dp[i]` 3. 踢劣质队尾并入队 `i`	若先压入 `i` 再查队头，会用 i 更新自己（全负子段和误得 0）
普通滑窗最值（包含当前位置 $[i-m+1, i]$）	1. 踢过期队头 2. 踢劣质队尾并入队 `i` 3. 查队头输出最值	若先查再入队，会漏掉当前元素 $a[i]$ 参与竞争

本页要点

DP 转移要求 $j < i$，因此本章 DP 题几乎全是“先踢过期 $\rightarrow$ 再查队头 $\rightarrow$ 最后入队”！

模板代码（1/2）：手写数组模拟队列（极致速度）

```
const int N = 200005;
int q[N], hh = 0, tt = -1;                  // hh 队头，tt 队尾；hh <= tt 表示非空
q[++tt] = 0;                                // 放入初始状态 0
for (int i = 1; i <= n; ++i) {
    while (hh <= tt && q[hh] < i - m) hh++; // 1. 踢出过期队头
    dp[i] = f[q[hh]] + g[i];                // 2. 队头即为最优前驱
    f[i] = dp[i];
    while (hh <= tt && f[q[tt]] >= f[i]) tt--; // 3. 踢出劣质队尾
    q[++tt] = i;                            // 4. 新决策入队
}
```

本页要点

竞赛里手写数组队列 `hh = 0, tt = -1` 比 `std::deque` 更快、常数更小，且不容易爆内存。

模板代码（2/2）：STL deque 写法对照

```
deque<int> q{0};                              // 初始化直接包含下标 0
for (int i = 1; i <= n; ++i) {
    while (!q.empty() && q.front() < i - m)
        q.pop_front();                        // 1. 踢过期队头
    dp[i] = f[q.front()] + g[i];              // 2. 查队头更新 dp[i]
    f[i] = dp[i];
    while (!q.empty() && f[q.back()] >= f[i])
        q.pop_back();                         // 3. 踢劣质队尾
    q.push_back(i);                           // 4. 新决策入队
}
```

本页要点

平时练习和讲义用 deque<int> q{0} 语义最清晰；卡常时一键换成数组 hh/tt 即可。

上机验证与反事实追问

3 组标准验证（以 $f[j]$ 滑窗最小值为例，$m = 3$）

1. 单调递增序列：$f = [0, 1, 2, 3, 4] \to$ 队列从不踢尾，长度最多达 3，各步最小分别为 $0, 0, 0, 1, 2$。
2. 单调递减序列：$f = [0, 5, 4, 3, 2] \to$ 除 0 外每次都会踢空前面，预期最后队列只剩 2。
3. 含相等元素：$f = [0, 2, 2, 2] \to$ 必须用 >= 踢掉旧的相等元素，让下标更大的 2 留下。

反事实追问

如果把“先查队头算 dp[i]，再把 i 入队”的顺序调反，在求“非空子段最大和”且序列全为负数 $[-5, -2, -8]$ 时会怎样？

$\Rightarrow$ 入队 i 时因为 $s[i]$ 比队头更小，会把前面全踢掉；接着查队头得到 s[i] - s[i] = 0，误以为最大子段和是 0（空段），导致 **WA**！

本页要点

“踢相等元素用 >=”和“先查后插”是保证队列最优与防空段的关键细节。

队列机制易错对照：WA / TLE / RE 定位

错误类型	常见错误写法	正确做法与原因
RE（空队列）	`while (q.front() < i - m)` 没判 `!q.empty()`	队列一旦空了再调用 `q.front()` 直接崩溃
WA（边界差 1）	`q.front() <= i - m`	允许距 i 恰好为 m 时只能踢 `< i - m`，多写等号会少一个合法决策
WA（比错方向）	求窗口最大值却写了 `>=`	求最小踢 `>=`（队内递增）；求最大踢 `<=`（队内递减）
RE（数组太小）	手写 `q[N]` 开得比 n 小	下标 $0 \sim n$ 共 $n+1$ 个元素都可能入队，数组至少开 $n+5$

本页要点

写完 `while` 循环，第一眼先看前面有没有 `!q.empty()`（或 `hh <= tt`）！

生活引入：每种物品最多拿 s 件，怎么不拆着算？

- 在第 1 章的背包模型里，0-1 背包每件物品只能拿 0 或 1 个；现在变成多重背包：第 i 种物品体积 v、价值 w，最多拿 s 件，背包总容量 V。
- 朴素写法：枚举容量 j 从 V 到 0，再枚举拿 $k = 0 \dots s$ 件，复杂度 $O(n \cdot V \cdot s)$。
- 当 $n = 100, V = 2 \times 10^4, s = 1000$ 时，运算量高达 2×10^9，直接 **TLE**！
- 二进制拆分能降到 $O(nV \log s)$，但单调队列还能再砍掉 $\log s$，直接做到极限的 $O(nV)$！

本页要点

用单调队列优化多重背包，可以把复杂度的 $\log s$ 因子彻底抹掉，达到理论最优 $O(nV)$！

新术语大白话：按余数 r = j % v 分组

核心观察：不同余数之间根本不会互相转移！

假设当前物品体积是 $v=3$。容量 $j=7$ 只能从 $7-3=4$、$7-6=1$ 转移过来；它永远不可能从容量 $2,3,5,6$ 转移过来！

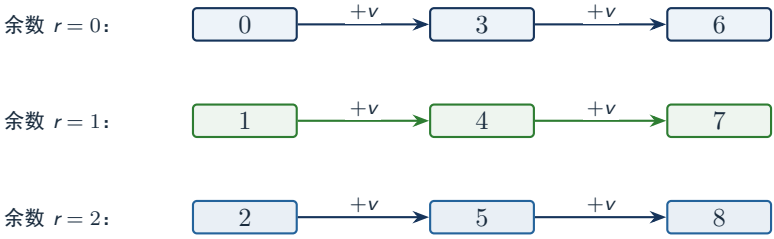
- 把所有容量 $0 \sim V$ 按照除以 v 的余数 $r \in [0, v-1]$ 分成 v 条完全独立的“流水线”：
  - 第 r 组只包含：$r,\ r+v,\ r+2v,\ \ldots,\ r+kv$。
- 在同一条流水线上，设当前容量是 $r+k\cdot v$，最多拿 s 件物品，则它只能从前驱 $r+p\cdot v$（其中 $k-s \le p \le k$）转移过来：

$$dp[r+kv] = \max_{k-s\le p\le k}\{g[r+pv]+(k-p)w\} = \max_{k-s\le p\le k}\{g[r+pv]-p\cdot w\}+k\cdot w$$

本页要点

把 $(k-p)w$ 拆成只含 p 的 $-p\cdot w$ 和只含 k 的 $+k\cdot w$，瞬间就变成了标准滑窗！

图解：按余数分组的独立流水线（以 $v=3, V=8$ 为例）



本页要点

每组内部下标步长为 v，相当于在“压缩后的数组”上跑宽度为 $s+1$ 的滑动窗口！

手算跟踪表：余数 $r=0$ 的单调队列推演（$v=2, w=3, s=2, V=6$）

小数据设定

上一层数组 `g = [0, 0, 1, 0, 5, 0, 2]`。只看余数 $r=0$ 的容量 $0,2,4,6$（即 $k=0,1,2,3$），窗口最大跨度 $s=2$。

k	$j=2k$	入队值 $g[j]-k\cdot w$	踢过期后队列（最大）	新 `dp[j]` = 队头 $+kw$
0	0	$0-0=0$	`[0:0]`	$0+0=0$
1	2	$1-3=-2$	`[0:0, 1:-2]`	$0+3=3$（拿 1 件）
2	4	$5-6=-1$	踢 `1`，`[0:0, 2:-1]`	$0+6=6$（拿 2 件）
3	6	$2-9=-7$	踢 `0`，队头 `2:-1`	$-1+9=8$（旧 5+1 件）
止	$k=4$	退出循环	—	得 $[0,3,6,8]$

本页要点

注意入队顺序：因为允许“拿 0 件”（即 $p=k$ 从自己上一轮 $g[j]$ 继承），所以先入队 k 再取队头！

正确性论证：偏移量剥离与拷贝数组 g

阶段	等价不变量：在当前物品处理中，g[] 保存上一件物品结束时的旧状态
初始化	开始处理本物品前备份 g = dp，保证队列里比较的全是“未放入本物品”的旧值。
保持	枚举余数 r 及步数 k，踢掉 $p < k - s$ 的队头，用 g[r+kv] - k*w 维护单调递减队列，队头加 $k \cdot w$ 即 $\max_{k-s \le p \le k}(g[r+pv] + (k-p)w)$。
终止	所有 $r \in [0, v-1]$ 扫完恰好不重不漏覆盖全部容量 $0 \dots V$。

本页要点

为什么一定要备份 g = dp？因为窗口是正序 $k = 0, 1, 2 \dots$ 推的，若不备份就会把当前物品重复塞进自己（变成完全背包）！

模板代码（1/2）：外层物品循环与数组备份

```
vector<int> dp(V + 1, 0);
for (int idx = 0; idx < n; ++idx) {
    int v, w, s; cin >> v >> w >> s;        // 体积 v，价值 w，最多 s 件
    vector<int> g = dp;                     // 备份上一轮 DP 状态，防正序串行
    for (int r = 0; r < v; ++r) {           // 枚举余数 r = 0 .. v-1
        deque<int> q;                       // 每条余数流水线用独立单调递减队列
        // 内部循环按步长 v 推进：j = r, r+v, r+2v ...
```

本页要点

每一轮物品处理开始前做一次 `vector<int> g = dp;`，彻底隔离旧状态与新状态。

模板代码（2/2）：单余数流水线滑动窗口

```
        for (int k = 0; r + k * v <= V; ++k) {
            int j = r + k * v;              // 当前处理的背包容量
            while (!q.empty() && q.front() < k - s) q.pop_front();
            while (!q.empty() && g[r + q.back() * v] - q.back() * w <= g[j] - k * w
    )
                q.pop_back();               // 踢出“值不如新人”的旧决策
            q.push_back(k);                 // 允许拿 0 件（从 g[j] 继承），先入队
            int p = q.front();
            dp[j] = g[r + p * v] + (k - p) * w; // 取队头更新
        }
    }
}
```

本页要点

因为允许拿 0 件（$p = k$），这里必须是“先入队 k 再查队头”；求最大价值踢 <= 队尾。

上机验证与易错对照

3 组标准验证（V 容量下最大价值）

1. 退化为 **0-1** 背包：$s = 1 \to$ 每组最多隔 1 步，结果与 0-1 背包完全一致。
2. 手算测试组：$n = 1, V = 6, v = 2, w = 3, s = 2 \to$ 最多拿 2 件，预期输出 6。
3. 装不满边界：$V = 5, v = 6, w = 10, s = 3 \to$ 放不进任何一件，预期输出 0。

易错点	错误写法	正确做法
WA（忘备份）	直接在 dp 上边读边改	必须备份 g = dp，队列里读取 g[] 写入 dp[]
WA（队列不清空）	把 deque<int> q; 写在 for r 外面	每条余数流水线必须使用全新的空队列！

本页要点

多重背包单调队列三大铁律：备份 g、按余数 r 开新队列、先入队 k 再取队头。

例 1 | [基础] 最大连续和（一本通 1598）： 先读懂要求

大白话题意

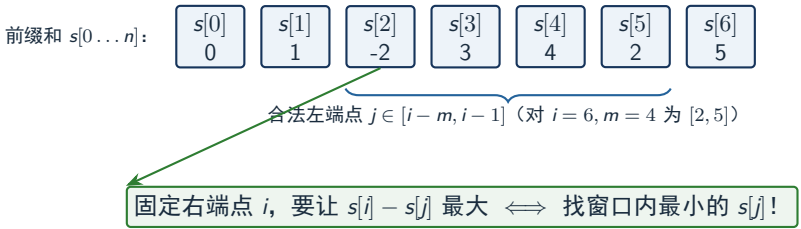
给一个长度为 n（$1 \le n \le 2 \times 10^5$）的整数序列（可能包含负数），从中找一段连续、非空且长度不超过 m 的子序列，使它的和最大。

项目	说明
输入	第一行 n, m；第二行 n 个整数（绝对值 < 1000）
处理	转化为前缀和 $\max_{i-m \le j < i}(s[i] - s[j]) = s[i] - \min_{i-m \le j < i} s[j]$
输出	一个整数，表示长度不超过 m 的最大连续子序列和
原题样例	`6 4` 与 `1 -3 5 1 -2 3` $\rightarrow$ 输出 `7`（选子段 `5 1 -2 3`）

本页要点

难度标签：**[基础]**。用前缀和把“区间和”变成两个前缀之差，瞬间套进单调队列模型！

例 1 | 最大连续和： 思路与建模图解



本页要点

要求子段非空，所以 j 严格小于 i（不能取 $j=i$）；队列维护的是递增的 $s[j]$！

例 1 | 最大连续和： 样例手推跟踪表

样例：$n=6, m=4$，$A=[1,-3,5,1,-2,3]$，前缀和 $s=[0,1,-2,3,4,2,5]$

i	$s[i]$	踢过期后队头	$s[i]-s[\text{front}]$	当前 ans	入队后队列（下标:s）
初态	$s[0]=0$	—	—	$-\infty$	`[0:0]`
1	1	`0:0`	$1-0=1$	1	`[0:0, 1:1]`
2	-2	`0:0`	$-2-0=-2$	1	`[2:-2]`（踢光 $0,1$）
3	3	`2:-2`	$3-(-2)=5$	5	`[2:-2, 3:3]`
4	4	`2:-2`	$4-(-2)=6$	6	`[2:-2, 3:3, 4:4]`
5	2	`2:-2`	$2-(-2)=4$	6	`[2:-2, 5:2]`（踢 $3,4$）
6（止）	5	`2:-2`	$5-(-2)=7$	**7**	`[2:-2, 5:2, 6:5]`

本页要点

当 $i=2$ 时 $s[2]=-2$ 把前面全踢空，所以从 $i=3$ 开始队头一直是 -2，最终在 $i=6$ 算出 $5-(-2)=7$！

例 1 | 最大连续和： 核心步骤与模板改动

按顺序计算	纸上得到	核对问题
1. 读入并求前缀和 s[i]	$s[0]=0, s[i]=s[i-1]+x$	序列和是否用了 long long?
2. deque<int> q{0}, ans=LLONG_MIN	放入 0，答案初值负无穷	全负数时初值设 0 会错吗？
3. 踢 q.front() < i - m	保证子段长度 $i-j \le m$	为什么不是 <= i - m?
4. 先用 s[i]-s[q.front()] 更新 ans，再踢尾入队	防止 i 入队后算出 $s[i]-s[i]=0$	序列全负时能选空子段吗？

相对基础模板改了哪里

基础模板把最优值存进 dp[i]；本题不依赖历史 dp[j]，而是直接拿前缀和 s[j] 进队，每步用 s[i] - s[q.front()] 打擂台更新全局 ans。

本页要点

记住：必须先算 ans = max(ans, s[i] - s[q.front()])，最后再把 i 压入队尾！

例 1 | 最大连续和： 核心代码（1/2）：前缀和与队列初始化

```cpp
int n, m; if (!(cin >> n >> m)) return 0;
vector<long long> s(n + 1, 0);
for (int i = 1; i <= n; ++i) {
    long long x; cin >> x;
    s[i] = s[i - 1] + x;                    // 预处理前缀和
}
deque<int> q{0};                            // 放入 s[0]=0 允许选从第 1 位开始的段
long long ans = LLONG_MIN;                  // 防止序列全负时误输出 0
```

本页要点

代码与 code/max_subarray_len.cpp 一致：ans 必须初始化为 LLONG_MIN！

例 1 | 最大连续和： 核心代码（2/2）：滑窗淘汰与更新

```
for (int i = 1; i <= n; ++i) {
    while (q.front() < i - m) q.pop_front(); // 1. 踢出长度超过 m 的过期起点
    ans = max(ans, s[i] - s[q.front()]);     // 2. 先更新答案，防止取到空段
    while (!q.empty() && s[q.back()] >= s[i]) q.pop_back(); // 3. 维护递增
    q.push_back(i);                          // 4. 当前下标入队
}
cout << ans << '\n';
```

本页要点

因为每次循环都会至少留着上一个 i-1（且 $m \geq 1$），队头永不空；先查后插杜绝全负误选空段。

例 1 | 最大连续和： 易错与 3 组自测边界

高频易错点

- 把 ans 初始化为 0：当输入全为负数（如 -5 -2 -8）时，正确答案是 -2 而不是 0（**WA**）。
- 先入队 i 再更新 ans：全负时 $s[i]$ 会踢空队列并成为队头，算出 $s[i]-s[i]=0$（**WA**）。

3 组自测数据与预期输出

1. 全负数边界：3 2 / -5 -2 -8 → 预期输出 -2。
2. 长度为 1 限制：4 1 / 3 -1 5 2 → 只能选单个元素，预期输出 5。
3. 原题样例：6 4 / 1 -3 5 1 -2 3 → 预期输出 7。

本页要点

测全负数 -5 -2 -8 是检验本题是否写对的最快试金石！

例 2 | [基础] 烽火传递（一本通 1602）： 先读懂要求

大白话题意

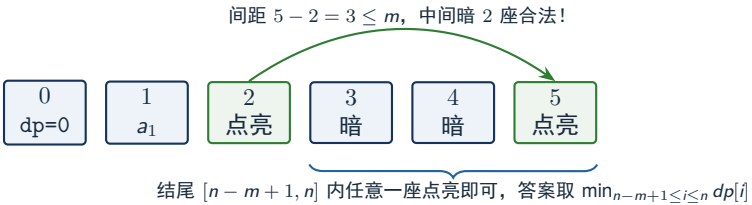
有 n 座烽火台（$1 \le n, m \le 2 \times 10^5$），点燃第 i 座需要代价 a_i。
要求任意连续 m 座烽火台里至少有 1 座发出信号，求准确传递情报的最小总代价。

项目	说明
输入	第一行 n, m；第二行 n 个整数 $a_1 \dots a_n$（$1 \le a_i \le 1000$）
处理	设 $dp[i]$ 为第 i 座必点且覆盖前 i 座的最小代价，$dp[i] = a_i + \min_{i-m \le j < i} dp[j]$
输出	一个整数，表示最小总代价
原题样例	5 3 与 1 2 5 6 2 $\rightarrow$ 输出 4（点燃第 $2, 5$ 座）

本页要点

难度标签：**[基础]**。标准的“自己转移自己”滑窗 DP，唯一要注意结尾不是输出 `dp[n]`！

例 2 | 烽火传递： 思路与建模图解



本页要点

因为第 n 座未必一定要点亮，只要最后 m 座里至少有一座亮着就覆盖完了全段！

例 2 | 烽火传递： 样例手推跟踪表

样例：$n=5, m=3$， $a=[1,2,5,6,2]$

i	$a[i]$	踢过期后队头	$dp[i]=dp[\text{front}]+a[i]$	入队后队列（下标:dp）
初态	—	—	$dp[0]=0$	`[0:0]`
1	1	`0:0`	$0+1=1$	`[0:0, 1:1]`
2	2	`0:0`	$0+2=2$	`[0:0, 1:1, 2:2]`
3	5	`0:0`	$0+5=5$	`[0:0, 1:1, 2:2, 3:5]`
4	6	踢 `0` → 队头 `1:1`	$1+6=7$	`[1:1, 2:2, 3:5, 4:7]`
5（止）	2	踢 `1` → 队头 `2:2`	$2+2=4$	`[2:2, 5:4]`（踢 $3,4$）

本页要点

最后从 $[5-3+1,5]=[3,5]$ 中取最小：$\min(dp[3],dp[4],dp[5])=\min(5,7,4)=4$！

例 2 | 烽火传递： 核心步骤与模板改动

按顺序计算	纸上得到	核对问题
1. `deque<int> q{0}, dp[0]=0`	前 m 座可直接从 0 转移	$i = m$ 时距 0 恰好 m 座合法吗？
2. 踢 `q.front() < i - m`	相距超过 m 会漏掉中间一段	为什么不能写成 `<= i - m`？
3. `dp[i] = dp[q.front()] + a[i]`	队头最小前驱加上当前代价	有没有在入队 `i` 之前计算？
4. 最后扫 `i = n-m+1 .. n`	取 `min(ans, dp[i])`	直接输出 `dp[n]` 会发生什么？

相对基础模板改了哪里

基础模板直接输出 `dp[n]`；本题最后一座烽火台不一定要点，因此循环结束后要补一个扫描最后 m 个状态取最小值的循环。

本页要点

覆盖型 DP 的三大特征：初态 $dp[0] = 0$、窗口宽度 m、答案在 $[n - m + 1, n]$ 取极值。

例 2 | 烽火传递： 核心代码（1/2）：读入与单调队列转移

```cpp
int n, m; if (!(cin >> n >> m)) return 0;
vector<long long> a(n + 1), dp(n + 1, 0);
for (int i = 1; i <= n; ++i) cin >> a[i];

deque<int> q{0};
for (int i = 1; i <= n; ++i) {
    while (!q.empty() && q.front() < i - m) q.pop_front(); // 距 i 超 m 则漏段
    dp[i] = dp[q.front()] + a[i];          // O(1) 取得合法区段内最优前驱
    while (!q.empty() && dp[q.back()] >= dp[i]) q.pop_back();
    q.push_back(i);
}
```

本页要点

代码与 `code/beacon_dp.cpp` 一致；$n = 2 \times 10^5, a_i = 1000$ 时开 `long long` 更稳妥。

例 2 | 烽火传递： 核心代码（2/2）：扫描末尾窗口

```
long long ans = LLONG_MAX;
for (int i = n - m + 1; i <= n; ++i) {    // 最后 m 座烽火台至少亮 1 座
    ans = min(ans, dp[i]);
}
cout << ans << '\n';
```

为什么不直接输出 dp[n]？

dp[i] 的含义是“点燃第 i 座且覆盖前 i 座”。最后一座不一定要点燃，只要 $[n-m+1, n]$ 内至少有一座点亮就覆盖完了整条路线！

本页要点

这是覆盖型单调队列 DP 最容易漏掉的收尾一步！

例 2 | 烽火传递： 易错与 3 组自测边界

高频易错点

- 直接输出 `dp[n]`：当最后一座代价极大（如 `5 3 / 1 2 5 6 100`）时，正确答案是只点第 3 座花费 5，输出 `dp[n]` 会错成 102（**WA**）。
- 过期条件写成 `<= i - m`：当 $i = m$ 时会把 0 踢掉，导致前 m 座里若选第 m 座就找不到前驱（**RE**）。

3 组自测数据与预期输出

1. 结尾代价极大：`5 3 / 1 2 5 6 100` $\rightarrow$ 选第 3 座，预期输出 5。
2. 全部必点边界：`3 1 / 4 5 6` $\rightarrow$ 每 1 座至少点 1 座，预期输出 15。
3. 只需点 1 座：`3 3 / 10 2 10` $\rightarrow$ 预期输出 2。

本页要点

“结尾扫后 m 座”几乎是所有区间覆盖 DP 的共同特征，务必牢记！

例 3 | [提高] 修剪草坪（一本通 1599）： 先读懂要求

大白话题意

有 N 只奶牛排成一排（$1 \le N \le 10^5$），第 i 只效率为 E_i（$0 \le E_i \le 10^9$）。
不能安排超过 K 只连续的奶牛工作（否则罢工），求能获得的最大总效率。

项目	说明
输入	第一行 N, K；接下来 N 行每行一个整数 E_i
处理	正难则反：最大化工作效率 $\iff$ 最小化“不选的奶牛”效率损失
输出	一个整数，表示最大总效率
原题样例	5 2 / 1 2 3 4 5 $\to$ 输出 12（舍弃第 3 只，得 $1+2+4+5=12$）

本页要点

难度标签：[提高]。直接求“选谁”状态难写，反过来求“让谁休息”立刻化归为《烽火传递》！

例 3 | 修剪草坪： 正难则反建模图解



总效率最大 = 全部效率总和 total − 最小休息损失lost

本页要点

令窗口宽度 $m = K+1$，把“休息”看成《烽火传递》里的“点亮”，代码几乎一模一样！

例 3 | 修剪草坪： 样例手推跟踪表

样例：$N=5, K=2 \Longrightarrow m=3$，$E=[1,2,3,4,5]$，总效率 total $=15$

i	$E[i]$	踢过期（$<i-3$）后队头	损失 $dp[i]=dp[\text{front}]+E[i]$	队列（下标:dp）
初态	—	—	$dp[0]=0$	`[0:0]`
1	1	`0:0`	$0+1=1$	`[0:0, 1:1]`
2	2	`0:0`	$0+2=2$	`[0:0, 1:1, 2:2]`
3	3	`0:0`	$0+3=3$	`[0:0, 1:1, 2:2, 3:3]`
4	4	踢 `0` → 队头 `1:1`	$1+4=5$	`[1:1, 2:2, 3:3, 4:5]`
5（止）	5	踢 `1` → 队头 `2:2`	$2+5=7$	`[2:2, 3:3, 5:7]`

本页要点

最后从 $[5-2,5]=[3,5]$ 取最小损失 lost $=\min(3,5,7)=3$，答案 $=15-3=12$！

例 3 | 修剪草坪： 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 E_i 顺便累加 `total`	总和可达 10^{14}	`total` 用了 `long long` 吗？
2. 设窗口 $m = k + 1$	相邻休息差 $\le k + 1$	为什么是 $k + 1$ 而不是 k？
3. 跑《烽火传递》单调队列	每只牛必休息的最小损失	队列初值放了 0 吗？
4. 扫 $[\max(0, n - k), n]$ 取 `lost`	输出 `total - lost`	$k \ge n$ 时能全选吗？

相对《烽火传递》改了哪里

最小休息间隔 $m = k + 1$； 求最大保留值，最后输出 `total - lost`； 扫描起点取 `max(0, n - k)`，兼容 $k \ge n$ 损失为 0 的情况。

本页要点

用 `max(0, n - k)` 作扫描起点，顺便处理了“一只都不休息（损失为 0）”的边界！

例 3 | 修剪草坪： 核心代码（1/2）：读入与窗口转换

```
int n, k; if (!(cin >> n >> k)) return 0;
long long total = 0;
vector<long long> e(n + 1), dp(n + 1, 0);
for (int i = 1; i <= n; ++i) {
    cin >> e[i];
    total += e[i];                          // 累加全部奶牛效率总和
}
int m = k + 1;                              // 每 k+1 只奶牛至少休息 1 只
deque<int> q{0};                            // dp[0]=0 允许前 k 只内首次休息
```

本页要点

代码与 code/trim_lawn.cpp 一致：total 累加必须在 long long 下进行防溢出。

例 3 | 修剪草坪： 核心代码（2/2）：求最小损失与作差

```
for (int i = 1; i <= n; ++i) {
    while (!q.empty() && q.front() < i - m) q.pop_front();
    dp[i] = dp[q.front()] + e[i];          // 跳过第 i 只的最小效率损失
    while (!q.empty() && dp[q.back()] >= dp[i]) q.pop_back();
    q.push_back(i);
}
long long lost = LLONG_MAX;
for (int i = max(0, n - k); i <= n; ++i) lost = min(lost, dp[i]);
cout << total - lost << '\n';
```

本页要点

“正难则反”让《修剪草坪》直接复用《烽火传递》的全部转移逻辑！

例 3 | 修剪草坪： 易错与 3 组自测边界

高频易错点

- 不开 `long long`：$N = 10^5, E_i = 10^9$ 时总和高达 10^{14}，用 `int` 必溢出（**WA**）。
- 把 m 错写成 k：连续最多 k 只工作，意味着连续 $k+1$ 只里必须有 1 只休息，所以 $m = k+1$（**WA**）。

3 组自测数据与预期输出

1. $k = n$（全选无限制）：`3 3 / 4 5 6` → 损失为 0，预期输出 `15`。
2. $k = 0$（一只也不能干）：`3 0 / 4 5 6` → 全部休息，预期输出 `0`。
3. 大数防溢出：`2 1 / 1000000000 1000000000` → 预期输出 `1000000000`。

本页要点

自测 $k = 0$ 输出 0、$k = n$ 输出全和，就能确认窗口和扫描下界 `max(0, n - k)` 完全正确。

例 4 | [提高] 绿色通道（一本通 1603）： 先读懂要求

大白话题意

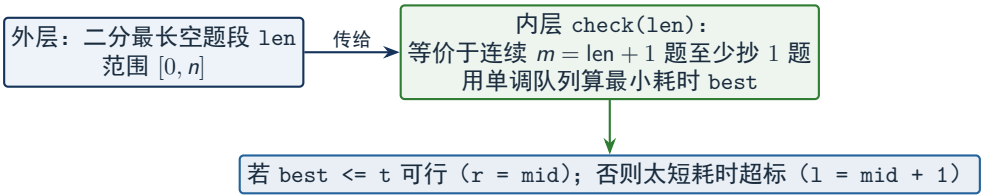
共有 n 道题（$n \le 5 \times 10^4$），抄第 i 题花 a_i 分钟，总时间不能超过 t 分钟。
未抄的连续题目组成“空题段”，老师只看最长空题段的长度。求在不超过 t 分钟的前提下，最长空题段长度最小是多少。

项目	说明
输入	第一行 n, t；第二行 n 个整数 $a_1 \dots a_n$
处理	二分“最长空题段长度 mid”，用单调队列检验最小耗时是否 $\le t$
输出	一个整数，表示最长空题段至少有多长
原题样例	17 11 与 6 4 5 2 ... 3 5 $\rightarrow$ 输出 3

本页要点

难度标签：[提高]。典型套路：“最大值最小” $\Longrightarrow$ 二分答案 + 单调队列检验！

例 4 | 绿色通道： 二分 + 单调队列建模图解



本页要点

单调性：允许的空题段越长，要抄的题就越少，所需最少时间就越小！因此完全满足二分单调性。

例 4 | 绿色通道： 样例手推（二分过程）

样例：$n = 17, t = 11$，手推二分窗口长度 mid

轮次	$[l, r]$	mid（$m = \text{mid} + 1$）	check(mid) 最小耗时	更新
1	$[0, 17]$	8（每 9 题做 1）	第 $9, 13$ 题耗时 $4 \le 11$	$r = 8$
2	$[0, 8]$	4（每 5 题做 1）	最小耗时 ≤ 11	$r = 4$
3	$[0, 4]$	2（每 3 题做 1）	最小耗时 > 11（超时）	$l = 3$
4	$[3, 4]$	3（每 4 题做 1）	最小耗时 $10 \le 11$（可行）	$r = 3$
止	$l = r = 3$	退出循环	—	输出 **3**

本页要点

单调队列每次检验只要 $O(n)$，外层二分只需 16 次，总复杂度 $O(n \log n)$ 秒过！

例 4 | 绿色通道： 核心步骤与模板改动

按顺序计算	纸上得到	核对问题
1. 二分下界 `l=0`, 上界 `r=n`	覆盖“全抄”到“全空”	t 足够大时能取 0 吗？
2. `check(len)` 设 $m = \text{len} + 1$	连续 m 题至少抄 1 题	为什么是 $\text{len} + 1$？
3. 跑《烽火传递》求最小耗时	返回 `best <= t`	结尾扫 $[n - \text{len}, n]$ 吗？
4. `check(mid)` 真则 `r = mid`	否则 `l = mid + 1`	最终输出 `l`

相对《烽火传递》改了哪里

把《烽火传递》封装成 `check(len)`（$m = \text{len} + 1$，结尾扫 `max(0, n - len) .. n`），外面套整数二分！

本页要点

空题段长度 `len` 对应窗口 $m = \text{len} + 1$。

例 4 | 绿色通道： 核心代码（1/2）：check 检验函数

```
auto check = [&](int len) -> bool {
    int m = len + 1;                          // 连续 m 道题中至少做 1 道
    vector<long long> dp(n + 1, 0); deque<int> q{0};
    for (int i = 1; i <= n; ++i) {
        while (!q.empty() && q.front() < i - m) q.pop_front();
        dp[i] = dp[q.front()] + a[i];
        while (!q.empty() && dp[q.back()] >= dp[i]) q.pop_back();
        q.push_back(i);
    }
    long long best = LLONG_MAX;
    for (int i = max(0, n - len); i <= n; ++i) best = min(best, dp[i]);
    return best <= t;
};
```

本页要点

代码与 code/green_channel.cpp 一致；每次调用 check 都会新建干净的 dp 和 q。

例 4 | 绿色通道： 核心代码（2/2）：外层整数二分

```cpp
int n; long long t; if (!(cin >> n >> t)) return 0;
vector<long long> a(n + 1);
for (int i = 1; i <= n; ++i) cin >> a[i];

// (此处接上面 check(len) 函数的定义)
int l = 0, r = n;                              // 最长空题段在 0 到 n 之间
while (l < r) {
    int mid = (l + r) / 2;
    if (check(mid)) r = mid;                   // 可行则尝试更小的空题段
    else l = mid + 1;                          // 超时则必须允许更长的空题段
}
cout << l << '\n';
```

本页要点

注意 l=0 时对应 $m = 1$（一题不空，全部都要抄），r=n 时对应全部空着耗时 0，边界严丝合缝。

例 4 | 绿色通道： 易错与 3 组自测边界

高频易错点

- 把 m 写成了 `len`：若允许空 0 题（`len=0`），$m=0$ 会导致所有题都能不抄或直接越界（**WA/RE**）。
- 二分下界设为 1：当 t 足够大能抄完所有题时，答案是 0，下界设为 1 会错（**WA**）。

3 组自测数据与预期输出

1. 全都能抄完：`3 10 / 2 3 4` $\rightarrow$ 总耗时 $9 \le 10$，预期输出 0。
2. 一题都抄不起：`3 1 / 5 6 7` $\rightarrow$ 只能全空，预期输出 3。
3. 原题样例：`17 11 / ...` $\rightarrow$ 预期输出 3。

本页要点

用“全都能抄完输出 0”和“一题都抄不起输出 n”测一下二分上下界，稳过！

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
最大连续和（1598）	先 `push_back(i)` 再算答案	全负时算出 $S[i]-S[i]=0$ 错选空段，必须先查后入队
最大连续和（1598）	`ans` 初值设为 0	序列全负时答案是负数，初值必为 `LLONG_MIN`
烽火传递（1602）	结尾直接输出 `dp[n]`	最后一座未必点燃，必须扫 $[n-m+1,n]$ 取 `min`
修剪草坪（1599）	窗口宽度错写成 k 而非 $k+1$	连续不超过 k 只，等价于每 $k+1$ 只至少休息 1 只
绿色通道（1603）	二分上界 `r` 漏写或 `check` 忘清空	每次 `check(mid)` 用新队列，上界设为 `n`

本页要点

单调队列 DP 90% 的 WA 都来自“忘记放 0”“先插后查”以及“结尾没扫最后窗口”。

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）	使用 `LLONG_MIN / LLONG_MAX` 需包含 `<climits>` 或 `<bits/stdc++.h>`。
WA（答案错误）	查三处： 队列初值有没有放 `0`； 是否先查队头再入队 `i`； 覆盖类 DP 结尾是否扫了后 m 个。
RE（运行错误）	`while (q.front() < i - m)` 前漏写 `!q.empty()`，或者手写数组队列空间开小了。
TLE（超时）	忘写队尾淘汰 `while` 导致队列退化；或二分套单调队列时上界写得过大。

本页要点

调试时先造一组全负数（测最大连续和）和一组末尾最小（测烽火传递），一眼就能抓出 Bug。

提交前检查清单（六必查）

1. 单调队列初始化时是否已放入 0（对应 $dp[0]=0$ 或前缀和 $S[0]=0$）？
2. 循环内部顺序是否严格为：踢过期队头 -> 查队头算 dp[i] -> 踢劣质队尾 -> 入队 i？（多重背包除外）
3. 求全负数序列极值时，ans 初值是否设为 LLONG_MIN 而非 0？
4. 覆盖类 DP（烽火传递/修剪草坪/绿色通道）结尾是否从 $[n-m+1,n]$ 扫一遍求最优？
5. 淘汰方向是否记对：求窗口最小用 >=（队内递增），求窗口最大用 <=（队内递减）？
6. 所有前缀和与 dp 数组是否开了 long long（如 $N\le 10^5, E_i\le 10^9$ 时总和达 10^{14}）？

本页要点

养成习惯：写完代码先默念“放 0、踢头、取值、踢尾、入队、扫尾”十二字口诀。

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：1598	最大连续和	[基础]	前缀和 + 单调递增队列，code/max_subarray_len.cpp
学校 OJ：1602	烽火传递	[基础]	覆盖型滑窗 DP + 结尾扫 m 项，code/beacon_dp.cpp
学校 OJ：1599	修剪草坪	[提高]	正难则反转化 + long long，code/trim_lawn.cpp
学校 OJ：1603	绿色通道	[提高]	二分答案 + 单调队列检验，code/green_channel.cpp

完成标准

能准确判断转移方程是否可分离出 $f[j]+g[i]$；熟练写出四步循环与多重背包余数分组；独立 AC 上述 4 道真题。

课后安排与下一章预告

- 复习：合上课件再推一遍为什么《修剪草坪》“连续不超过 k 只”等价于“每 $k+1$ 只至少休息 1 只”。
- 练习：独立复写 code/ 下四份源码，并用全负数序列自测《最大连续和》。
- 下一章预告：第 6 章《斜率优化 DP》——当转移方程里出现了逃不掉的交叉乘积项 $t_i \cdot c_j$，单调队列失效了怎么办？把方程改写成直线的斜率与截距，用平面上的“凸壳”继续把 $O(n^2)$ 砍成 $O(n)$！

本页要点

记住：无交叉项用本章单调队列，有交叉项就进入第 6 章斜率优化！

本章一句话

本章一句话

分离出只跟前驱有关的项，用“队头踢过期、队尾踢又老又差”的双端队列把 $O(nm)$ 压成均摊 $O(n)$。

谢谢！