斜率优化 DP

化交叉项为一次函数截距：用下凸壳与叉积免除精度误差

HGNU-ACM 实验室算法组

配套：代数拆解 $y = kx + b$ + 下凸壳单调维护 + 二分切点 + 3 道经典真题精讲





本章目录

从交叉乘积到一次函数

下凸壳几何直觉与队列维护

防爆技巧与非单调斜率二分

前三节从“单调队列为什么失效”讲起，手把手把交叉项拆成直线方程，画出下凸壳并学会用 `__int128` 交叉相乘与二分找切点。

本章目录（续）

三大经典真题精讲

常见错误、检查清单与练习安排

后两节精讲《任务安排 2》《玩具装箱》《任务安排 3》三道真题，并附上提交前检查清单与练习索引。

温故知新：单调队列为什么处理不了交叉项？

- 上一章《单调队列优化 DP》里，转移式长这样：$f[i] = w_i + \min_j\{f[j] + g_j\}$。
- 它的特点是：只和 i 有关的项与只和 j 有关的项完全分开，谁大谁小只取决于 j 自己！
- 可一旦方程里出现 $a_i \cdot b_j$ 这种同时包含 i 和 j 的乘积项（叫交叉项），j_1 和 j_2 谁更优就会随着 i 变化而翻转。
- 这时单调队列不能直接比较 $f[j_1]$ 和 $f[j_2]$，必须请出今天的工具——斜率优化。

本页要点

单调队列只管“各算各的”线性转移；遇到 $a_i \cdot b_j$ 交叉项，就要把它画成平面上的直线！

暴力做法与复杂度：为什么 $O(n^2)$ 会 TLE？

以《任务安排 2》为例

第 i 个任务结束时，前面 j 个任务已经完成，第 $j+1 \sim i$ 个任务合并成新的一批：

$$f[i] = \min_{0 \le j < i} \left\{ f[j] + t_i(c_i - c_j) + s(c_n - c_j) \right\}$$

- 暴力做法：外层枚举 $i = 1 \dots n$，内层枚举 $j = 0 \dots i-1$，一共要做 $\frac{n(n+1)}{2}$ 次转移。
- 当 $n = 10^4$ 时运算约 5×10^7 次勉强擦边；但当 $n = 3 \times 10^5$（《任务安排 3》）时，$\frac{n^2}{2} \approx 4.5 \times 10^{10}$ 次，远超 1 秒 10^8 次上限，必然 **TLE**！

本页要点

$O(n^2)$ 暴力在 $n \ge 5 \times 10^4$ 时必超时，必须把内层找最优 j 从 $O(n)$ 压到均摊 $O(1)$ 或 $O(\log n)$！

从手算小例子引入：3 个任务怎么分批？

先用 $n=3, s=1$ 的小数据手算一遍

设耗时 $T=[1,3,2]$（前缀和 $t=[0,1,4,6]$），费用系数 $C=[3,2,1]$（前缀和 $c=[0,3,5,6]$，总费用 $c_3=6$）。

- 算 $i=1$：只有 $j=0$ 可选，$f[1]=0+1\times(3-0)+1\times(6-0)=9$。
- 算 $i=2$：比较 $j=0$ 与 $j=1$：
  - $j=0$：$0+4\times(5-0)+1\times(6-0)=26$；
  - $j=1$：$9+4\times(5-3)+1\times(6-3)=20$（更优，$f[2]=20$）。
- 算 $i=3$：比较 $j=0,1,2$，得 $j=2$ 最优，
  $f[3]=20+6\times(6-5)+1\times(6-5)=27$。

本页要点

手算发现：算 $f[i]$ 时，所有历史决策 j 的 $f[j]$ 和 c_j 早就是已知常数，只有 t_i 在变！

新术语大白话：决策点、斜率与截距

- 决策点 (x_j, y_j)：把过去每一个可选择的断点 j，变成平面直角坐标系上的一个固定点；它的横坐标 x_j 和纵坐标 y_j 只由 j 决定。
- 查询斜率 k_i：当前正在计算第 i 步时，把所有只和 i 有关、且乘在 x_j 前面的系数打包叫做斜率 k_i。
- 截距 b_i：斜率为 k_i 的直线穿过点 (x_j, y_j) 时，它在 y 轴上的交点高度 $b_i = y_j - k_i x_j$。
- 核心等价：原方程求最小的 $f[i]$，去掉纯含 i 的常数项后，就等价于找一个点 (x_j, y_j) 让截距 b_i 最小！

本页要点

把抽象的 DP 转移看成“在平面点集里，用斜率为 k_i 的直线去碰最底下的点”！

代数拆解三步法：把 $i \cdot j$ 变成 $y_j = k_i x_j + b_i$

拆解《任务安排》转移式的三步套路

原式：$f[i] = \min_{0 \le j < i}\{f[j] + t_i c_i - t_i c_j + sc_n - sc_j\}$。

1. 第一步（合并交叉项）：把同时含 i 和 j 的项凑在一起：$-(t_i + s) \cdot c_j$。
2. 第二步（纯 i 提出来）：把只含 i（或常数 c_n）的项提到 min 外面：$t_i c_i + sc_n$。
3. 第三步（对号入座）：令 $\underbrace{y_j}_{f[j]} - \underbrace{k_i}_{t_i+s} \cdot \underbrace{x_j}_{c_j} = b_i$，即 $y_j = k_i x_j + b_i$！

$$f[i] = t_i c_i + sc_n + \min_{0 \le j < i}\{y_j - k_i x_j\}$$

本页要点

记住口诀：“纯 j 做 y_j，交叉项拆出 $k_i \cdot x_j$，纯 i 丢到 min 外面”！

动手拆一拆：两道经典题的 x_j, y_j, k_i 对照表

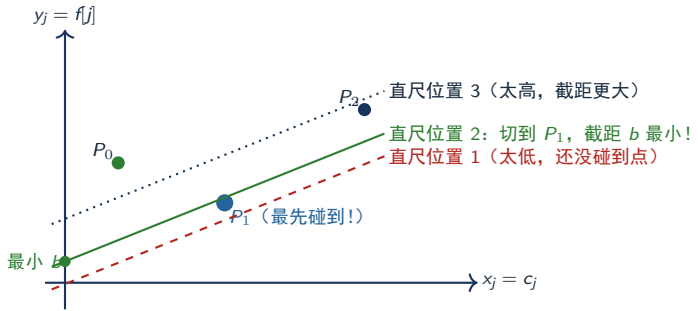
题目	转移方程核心项	决策点 (x_j, y_j)	查询斜率 k_i
任务安排 2	$f[j] - (t_i + s)c_j$	$x_j = c_j,\ y_j = f[j]$	$k_i = t_i + s$
打印文章	$f[j] + (S_i - S_j)^2 + M$	$x_j = S_j,\ y_j = f[j] + S_j^2$	$k_i = 2S_i$
玩具装箱	$f[j] + (P_i - P_j - L')^2$	$x_j = P_j,\ y_j = f[j] + P_j^2$	$k_i = 2(P_i - L')$

- 注意《打印文章》与《玩具装箱》里把平方展开：$(A_i - B_j)^2 = A_i^2 - 2A_iB_j + B_j^2$。
- 其中 B_j^2 只和 j 有关，必须老老实实放进 y_j 里；A_i^2 只和 i 有关，提到 min 外面！

本页要点

平方展开后的 B_j^2 属于纯 j 项，千万别漏写进 y_j 里！

TikZ 图解：截距最小化就是“斜率直尺从下往上碰到的第一个点”



本页要点

把一把固定斜率为 k_i 的直尺从无限下方往上平移，第一个碰到的点就对应最小的截距 b！

为什么单调队列失效了？决策优劣随 k_i 动态翻转

- 假设有两个决策点 $P_0(0,0)$ 和 $P_1(3,9)$，它们连线的斜率是 $\frac{9-0}{3-0}=3$。
- 当斜率 $k_i=2<3$ 时：$b_0=0-2\times 0=0$，$b_1=9-2\times 3=3$，P_0 更优！
- 当斜率 $k_i=5>3$ 时：$b_0=0-5\times 0=0$，$b_1=9-5\times 3=-6$，P_1 反而更优！
- 结论：点 j_1 和 j_2 没有绝对的“谁永远比谁好”，谁好取决于当前斜率 k_i 与两点连线斜率的比较！

本页要点

当 k_i 跨过两点连线的斜率时，最优决策就从左边的点“交接”给了右边的点！

符号与方向核对：求 min 看下凸壳，求 max 看上凸壳

优化目标	平移直尺方向	几何形状	连续三点 A, B, D（x 递增）
求最小值 min	从下往上平移（最小截距）	下凸壳	$slope(A, B) < slope(B, D)$
求最大值 max	从上往下平移（最大截距）	上凸壳	$slope(A, B) > slope(B, D)$

- 记忆技巧：求 min 找碗底，斜率越往右越陡（递增），所以叫下凸壳；
- 求 max 找拱顶，斜率越往右越缓（递减），所以叫上凸壳。
- 本章三道例题全都是求最小费用，因此重点吃透下凸壳。

本页要点

先看转移式是 min 还是 max，再决定维护下凸壳（斜率增）还是上凸壳（斜率减）！

本节易错：拆式子时把 k_i 和 x_j 搞混

常见错误写法

- 把 $y_j = k_i x_j + b_i$ 写成 $y_j = x_j k_i$ 却误把 t_i 当成 x_j；
- 移项时把 $-t_i c_j$ 的负号弄反，导致 $k_i = -(t_i + s)$；
- 展开 $(P_i - P_j)^2$ 忘了 P_j^2 项。

正确核对口诀

- 统一写成目标式：$b_i = y_j - k_i x_j$；
- x_j 和 y_j 里面绝对不能出现 i；
- k_i 里面绝对不能出现 j；
- 代回原式检验是否完全相等！（会触发 **WA**）

本页要点

写代码前一定在草稿纸上写一遍 $y_j - k_i x_j$，确认“x_j, y_j 只含 j，k_i 只含 i”！

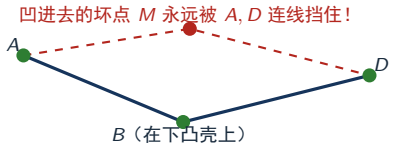
什么是下凸壳？大白话：像碗底一样向下凸的折线

- 把一堆已知决策点 (x_j, y_j) 按横坐标 x_j 从小到大排在平面上。
- 用一根橡皮筋从最下面兜住所有点，紧绷后的折线就叫做下凸壳。
- 下凸壳有一个神奇性质：沿着折线从左往右走，每一段的斜率严格递增（越往右越陡峭）。
- 更关键的是：无论查询斜率 k_i 等于多少，直尺从下往上平移时最先碰到的点一定在下凸壳的顶点上！

本页要点

下凸壳就像一根从下方兜住所有点的橡皮筋，不在橡皮筋上的点永远不可能最先被直尺碰到！

TikZ 图解：什么点必须从下凸壳淘汰掉？



- 看红虚线上的点 M：线段 AM 的斜率大于线段 MD 的斜率（先上坡后下坡，形成“上凸”小包）。
- 无论直尺斜率是多少，从下往上平移时，一定先碰到 A 或 D（或更低的 B），绝对轮不到被挡在上方三角形里的 M！

本页要点

只要出现 $slope(A, M) \geq slope(M, D)$，中间点 M 就是“凹进去的坏点”，直接淘汰！

三点共线怎么办？斜率相等时中间点也能删

- 如果三点 A, B, D 恰好在同一条直线上，即 $slope(A, B) = slope(B, D)$：
- 当直尺斜率恰好等于这条直线斜率时，A, B, D 会同时被碰到，选 A 或 D 的截距和选 B 一模一样！
- 当直尺斜率更小时先碰到 A，更大时先碰到 D，永远不需要专门保留 B。
- 因此淘汰条件写成大于等于（$slope(A, B) \geq slope(B, D)$ 就删 B），还能顺便把共线冗余点清掉。

本页要点

用 $\geq$ 淘汰队尾不仅能保证斜率严格递增，还能避免三点共线时凸壳越攒越长！

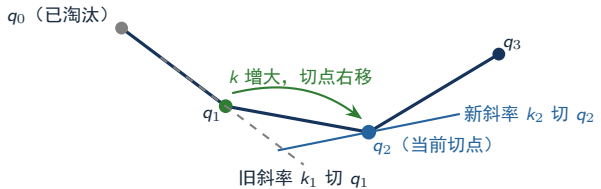
队头淘汰条件：当 k_i 单调递增时，旧队头永久出局

- 假设本题中查询斜率 k_i 随着 i 增大而单调递增（例如《任务安排 2》中 $k_i = t_i + s$，因为 $T_i > 0$）。
- 观察队头前两个点 q_0, q_1：如果 $slope(q_0, q_1) \le k_i$，等价于 $val(q_1, k_i) \le val(q_0, k_i)$（$q_1$ 的截距不比 q_0 差）。
- 因为以后所有的 $k_{i'}$ 只会比现在的 k_i 更大，所以 q_0 以后也永远赢不了 q_1！
- 于是放心执行 `hull.pop_front()`，把 q_0 永久丢出队列！

本页要点

只有当 k_i 单调递增时，队头落败的点才永远没有翻身机会，可以放心 `pop_front()`！

TikZ 图解：斜率 k_i 变大时，最优切点沿下凸壳向右滚动



本页要点

每个点最多进队一次、出队一次，所以维护下凸壳的均摊时间复杂度只有 $O(1)$！

队尾维护条件：新点加入前先踢掉不再下凸的队尾

- 算完 $f[i]$ 后，新的决策点 i 坐标为 (x_i, y_i)，准备压入凸壳队尾。
- 假设新点横坐标满足 x_i 单调不减（在所有前置和题目中都成立）。
- 只要队列里至少有 2 个点，且倒数第二个点 a、队尾点 b 与新点 i 满足：

$$slope(a, b) \geq slope(b, i)$$

  就说明 b 被新点 i 垫高凹进去了，立即 `hull.pop_back()`！
- 循环结束后，再执行 `hull.push_back(i)`。

本页要点

“先用队头算答案，再修剪队尾保下凸，最后把 i 放进队尾”——顺序千万不能反！

手算跟踪表（一）：4 个任务前 3 步的凸壳演变

小数据设定：$n=4, s=1$，$T=[1,3,2,2]$（$t=[0,1,4,6,8]$），$C=[3,2,1,2]$（$c=[0,3,5,6,8]$, $c_4=8$）

公式：$f[i]=t_i c_i + s c_4 + \min_j\{f[j]-(t_i+1)c_j\}$，初始 `hull = [0]`，$f[0]=0$。

i	k_i	队头比较与弹出	最优 j	$f[i]$	维护后 `hull`
1	2	仅 1 个点，不弹	0	$1\times3+8+0=11$	`[0, 1]`
2	5	$val(1,5)=-4\le val(0,5)=0$，弹 0	1	$20+8-4=24$	`[1, 2]`
3	7	$val(2,7)=-11\le val(1,7)=-10$，弹 1	2	$36+8-11=33$	`[2, 3]`

本页要点

注意第 2 步：0 被弹掉后，队列里只有 1 和新加入的 2，两点连线斜率是 $(24-11)/(5-3)=6.5$。

手算跟踪表（二）：第 4 步与循环终止状态

i	k_i	队头比较与弹出	最优 j	$f[i]$	维护后 `hull`
3	7	已算完：$f[3]=33$	2	33	`[2, 3]`
4	9	$val(3,9)=-21 \le val(2,9)=-21$，弹 2	3	$64+8-21=51$	`[3, 4]`
终止	$i=5>n$，循环结束！输出 $f[4]=51$。				

- 看看第 4 步：$val(3,9) == val(2,9) == -21$，因为写了 <=，所以会把 2 弹掉换成 3，结果同样正确！
- 这组数据如果用 $O(n^2)$ 暴力检查所有 j，得到的最小值也正好是 51。

本页要点

手算跟踪表一定要把最后 $i > n$ 的“循环终止”那一行写出来，确认答案取自 $f[n]$！

为什么是对的？下凸壳不变量（初始化 / 保持 / 终止）

阶段	凸壳不变量（Loop Invariant）
初始化	$i = 1$ 前，`hull = {0}` 只含 (x_0, y_0)，天然是一个下凸壳，且包含当前所有合法决策。
保持	每轮： 弹掉的队头 q_0 满足 $slope(q_0, q_1) \le k_i \le k_{i'}$，此后永远劣于 q_1； 当前队头是下凸壳上首个斜率 $> k_i$ 段左端点，截距必最小； 弹掉的队尾 b 凹在 a, i 上方，永远不是最优切点。
终止	$i = n + 1$ 退出循环时，$f[1 \dots n]$ 全部由各自全局最优前驱算出，故 $f[n]$ 必为全局最优解。

本页要点

之所以能从 $O(n^2)$ 降到 $O(n)$，核心在于“被淘汰的点永远不可能在未来翻盘”！

单调斜率优化模板代码（1/2）：初始化与队头淘汰

```
auto val = [&](int j, long long x) {                    // 截距 b = y_j - k_i * x_j
    return f[j] - x * c[j];
};
auto bad = [&](int a, int b, int d) {                   // 叉积比较 slope(a,b) >= slope
    (b,d)
    return (__int128)(f[b] - f[a]) * (c[d] - c[b])
        >= (__int128)(f[d] - f[b]) * (c[b] - c[a]);
};
deque<int> hull{0};                                     // 务必先把初始决策点 0 放进队列
for (int i = 1; i <= n; ++i) {
    long long x = t[i] + s;                             // 当前查询斜率 k_i
    while (hull.size() > 1 && val(hull[1], x) <= val(hull[0], x))
        hull.pop_front();                               // 队头不如第二个点，永久淘汰
```

本页要点

先写好 val 和 bad 两个匿名函数（Lambda），主循环就会非常干净易读！

单调斜率优化模板代码（2/2）：转移计算与队尾维护

```
    f[i] = t[i] * c[i] + s * c[n] + val(hull.front(), x); // 用当前最优队头算出 f[i
    ]
    while (hull.size() > 1 && bad(hull[hull.size() - 2], hull.back(), i))
        hull.pop_back();                                  // 队尾凹点不满足下凸性，弹出
    hull.push_back(i);                                    // 把新决策点 i 压入凸壳队尾
}
```

为什么这样写

先算 $f[i]$ 再把 i 入队，因为第 i 批至少要包含第 i 个任务，不能自己转移给自己。

易错提醒

`hull.size() > 1` 决不能漏，否则只剩 1 个点时访问 `hull[1]` 会越界 **RE**！

本页要点

记住四步顺序：弹队头 $\rightarrow$ 算 $f[i]$ $\rightarrow$ 弹队尾 $\rightarrow$ 入队 i！

上机验证：3 组测试数据与反事实追问

测试场景	输入（n, s 与 (T_i, C_i)）	预期输出	考察目的
普通样例	`5 1 / 1 3, 3 2, 4 3, 2 3, 1 4`	153	多批次正常切分
最小边界	`1 5 / 3 4`	32	仅 1 个任务，$(5+3)\times 4=32$
特殊 $S=0$	`3 0 / 1 2, 2 3, 3 1`	21	无启动费时全部单独分批更优

反事实追问

- 问：如果把 `deque<int> hull{0};` 改成空队列 `deque<int> hull;` 会怎样？
- 答：$i=1$ 时直接访问 `hull.front()` 发生空队列越界，轻则 **WA**，重则当场 **RE**（段错误）！

本页要点

写完模板先测“$n=1$”和“$S=0$”两组极小边界，能一秒揪出初值漏放 0 的 Bug！

本节易错：忘记放初始点 0 与 hull.size() > 1 越界 RE

错误写法

- `deque<int> hull;`（忘了放 0）→ **RE**；
- `while (!hull.empty() && val(hull[1], x) ...)` → 仅剩 1 个点时访问 `hull[1]` 越界 **RE**；
- 先 `hull.push_back(i)` 再算 $f[i]$ → 允许空批次 **WA**。

正确写法

- 一律写成 `deque<int> hull{0};`；
- 弹队头与弹队尾的 `while` 条件第一项必须是 `hull.size() > 1`；
- 先算 $f[i]$，最后再 `hull.push_back(i)`。

本页要点

单调队列写斜率优化时，两个 `while` 循环的保护条件永远是 `hull.size() > 1`！

为什么不能用 double 算斜率？（除零与精度灾难）

- 很多同学初学斜率优化时，习惯写一个函数：`double slope(int a, int b) { return (double)(y[b]-y[a])/(x[b]-x[a]); }`。
- 这会埋下两颗致命定时炸弹：
  1. 除零崩溃：如果某个任务费用系数 $C_i = 0$，就会出现 $x_b = x_a$，分母为 0 算出 `inf` 或 `NaN`，比较大小直接乱套 **WA**！
  2. 精度丢失：`double` 只有 $15 \sim 16$ 位有效数字；当 $y_b \sim 10^{18}$ 时，末几位直接被四舍五入抹掉，斜率差 10^{-16} 就会错判！

本页要点

在整数算法题里，能用整数比较大小就绝对不用 `double` 除法！

整数交叉相乘：把斜率不等式变成乘法

不等式变形（假设 $x_a \le x_b \le x_d$，分母均非负）

我们要判断的其实是：

$$\frac{y_b - y_a}{x_b - x_a} \ge \frac{y_d - y_b}{x_d - x_b}$$

两边同时乘以非负整数 $(x_b - x_a)(x_d - x_b)$，不等号方向不变：

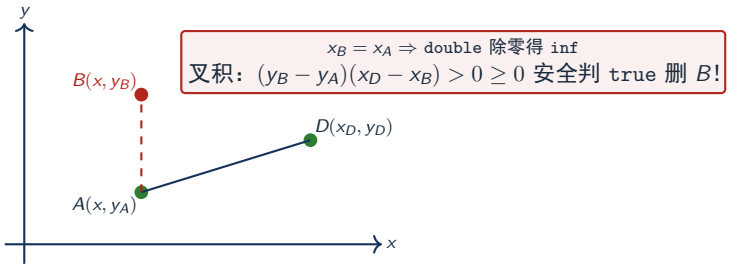
$$(y_b - y_a)(x_d - x_b) \ge (y_d - y_b)(x_b - x_a)$$

- 完全没有除法！就算 $x_b = x_a$，右边乘积自动变成 0，程序也不会崩溃。

本页要点

斜率不等式交叉相乘：左边分子 $\times$ 右边分母 $\ge$ 右边分子 $\times$ 左边分母！

TikZ 图解：横坐标相等（竖直段）时 double 崩溃 vs 叉积安全



本页要点

当两点横坐标相同时，上面那个点 B 永远比下面那个点 A 差，叉积含 >= 会自动把 B 删掉！

为什么必须用 __int128？$10^{12} \times 10^6$ 溢出 long long

- 来算一笔账：在《任务安排》或《玩具装箱》中，$f[i]$ 的值常常达到 $10^{12} \sim 10^{15}$，横坐标 c_i 达到 $10^6 \sim 10^7$。
- 交叉相乘时，两项乘积最大可达：

$$10^{14} \times 10^6 = 10^{20}$$

- 而 64 位整数 long long 的最大值只有约 9.22×10^{18}！一旦溢出就会变成负数，大小比较彻底反转 **WA**。
- 64 位竞赛环境下（GCC / Clang）自带 128 位整数 __int128，最大可达 1.7×10^{38}，装下 10^{20} 绰绰有余！

本页要点

叉积两边强转（__int128）只需多打 11 个字符，就能彻底消灭 90% 的隐形溢出 WA！

负分母陷阱：横坐标不单调时交叉相乘必须变号

- 前面我们能直接把 $\frac{y_b-y_a}{x_b-x_a} \geq \frac{y_d-y_b}{x_d-x_b}$ 变成乘法而不改符号，前提是两个分母 $(x_b - x_a)$ 和 $(x_d - x_b)$ 都大于等于 0。
- 如果遇到某道题，其中一个分母是负数（比如从右往左转移导致 x 递减），不等式两边同乘一个负数时，不等号必须反向！
- 因此，写交叉相乘前先确认两件事： x_j 是递增还是递减； 两个分母乘积的正负号。
- 也可以把原式整理成让 x_j 单调递增的形式（最稳妥、最不容易写错）。

本页要点

分母同号不变号，异号要变号；做题时尽量把横坐标 x_j 凑成单调不减！

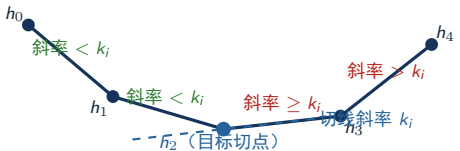
如果查询斜率 k_i 不单调怎么办？

- 在《任务安排 2》中，任务耗时 $T_i > 0$，所以 $k_i = t_i + s$ 单调递增，队头老点永不回头，可以放心 `pop_front()`。
- 但在《任务安排 3》中，允许 $T_i < 0$（任务耗时可正可负）！这就导致前缀和 t_i 会忽大忽小，查询斜率 k_i 不再单调！
- 如果还用 `pop_front()`，刚才在 $k = 10$ 时被淘汰的左端点，下一轮 k 跌回 2 时原本又是最优的，却已经被你删掉了！
- 解决办法：队头一个都不删，用 `vector` 保留整条下凸壳；查询时因为下凸壳斜率单调递增，直接在凸壳上二分查找切点！

本页要点

斜率不单调时：队头不删、队尾照常维护下凸壳，查询从均摊 $O(1)$ 变成二分 $O(\log n)$！

TikZ 图解：在完整下凸壳上二分查找切点



本页要点

下凸壳上相邻两点的截距差 $val(h_{m+1}, k_i) - val(h_m, k_i)$ 先负后正，满足单调性，可直接二分！

二分找切点手算跟踪表（5 点凸壳 + 两次倒退斜率查询）

假设凸壳存有 5 个点：$(0,0),(2,1),(4,4),(6,10),(8,20)$

相邻线段斜率依次为：$0.5,\ 1.5,\ 3.0,\ 5.0$（严格递增）。

轮次	k_i	二分区间 $[l,r]$	比较 $val(h_{mid},k_i)$ 与 $val(h_{mid+1},k_i)$	切点
1	3.5	$[0,4]\to mid=2$	$-10>-11$（右移）	$h_3(6,10)$
2	1.0	$[0,4]\to[0,2]$	h_0 斜率 $0.5<1.0<1.5$	$h_1(2,1)$
终止	第 2 次斜率从 3.5 倒退回 1.0，二分成功找回旧点 h_1！			

本页要点

二分找的是“第一个满足 $val(h_{mid},k_i)\le val(h_{mid+1},k_i)$ 的下标”，即 $l=r$ 处！

非单调斜率二分模板代码（1/2）：二分找切点

```
vector<int> hull{0};                                    // 队头不删，用 vector 存完整下
    凸壳
for (int i = 1; i <= n; ++i) {
    long long x = t[i] + s;                             // 斜率 x 可能忽大忽小，不能
    pop_front
    int l = 0, r = (int)hull.size() - 1;
    while (l < r) {
        int mid = (l + r) >> 1;
        if (val(hull[mid], x) <= val(hull[mid + 1], x)) r = mid;
        else l = mid + 1;                               // 右边点截距更小，往右半边找
    }
```

本页要点

当 `l < r` 时，`mid = (l + r) >> 1` 严格小于 `r`，所以 `hull[mid + 1]` 绝对不会越界！

非单调斜率二分模板代码（2/2）：转移与队尾维护

```
    f[i] = (long long)(Wide(t[i]) * c[i] + Wide(s) * c[n] + val(hull[l], x));
    while (hull.size() > 1 && bad(hull[hull.size() - 2], hull.back(), i))
        hull.pop_back();                                  // 队尾照样维护下凸壳
    hull.push_back(i);
}
```

本页要点

横坐标依然单调不减，所以队尾维护与单调队列版完全相同，时间复杂度 $O(n \log n)$！

上机验证：含负数耗时与相等 c_j 的 3 组验证

测试场景	输入（n, s 与 (T_i, C_i)）	预期输出	考察目的
含负耗时	`3 2 / 5 1, -3 1, 4 1`	21	斜率倒退时二分找回旧点
零费用系数	`3 1 / 2 0, 3 0, 1 2`	14	c_j 相等时不触发除零崩溃
大数防爆	$N = 10^4, T_i = 200, C_i = 200$	无负数溢出	验证 `__int128` 叉积稳定性

反事实追问

- 问：如果把 `bad()` 里的 `Wide(...)` 删掉改回普通 `long long`，上面哪组数据最容易出错？
- 答：第 3 组大数数据！因为 $f[i]$ 可达 10^{14}、$c[i]$ 可达 2×10^6，乘积突破 10^{19} 溢出成负数，把好点全踢掉导致 **WA**！

本页要点

只要数据里 $f[i] \times c[i]$ 有机会超过 9×10^{18}，就必须把 `val` 和 `bad` 全放进 `__int128`！

本节易错：非单调斜率照抄 pop_front 导致 WA

错误写法

- 见斜率优化就无脑写 deque 弹队头（忽视 $T_i < 0$）；
- 用 double 算斜率被 $C_i = 0$ 除零坑害；
- 横坐标 x_j 也不单调时直接插队尾（此时需要 CDQ 分治或李超线段树）。

正确决策流程

- 第一步：看 x_j 是否单调？单调才能直接插队尾；
- 第二步：看 k_i 是否单调？单调就 pop_front（$O(n)$），不单调就凸壳二分（$O(n \log n)$）；
- 第三步：一律用 __int128 交叉相乘。

本页要点

做斜率优化前先问自己两个单调性问题：横坐标 x_j 单调吗？查询斜率 k_i 单调吗？

例 1 | 任务安排 2（1607）：先读懂要求 [基础]

- 输入：第一行 N（$1 \le N \le 10^4$），第二行启动时间 S（$0 \le S \le 50$）；接下来 N 行每行正整数 T_i, C_i（$1 \le T_i, C_i \le 100$）。
- 处理：把 N 个任务按顺序分成若干连续批次；每批启动耗时 S 再加该批所有 T_k，同批任务在批末同一时刻完成，费用为完成时刻 $\times C_i$。
- 输出：最小总费用。

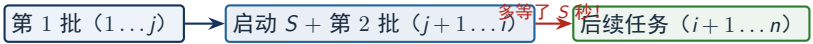
原题样例

输入：5 / 1 / 1 3, 3 2, 4 3, 2 3, 1 4 $\Rightarrow$ 输出：153

本页要点

先别急着写代码，先确认：同一批里每个任务的完成时刻都等于这批结束的时刻！

例 1 | 怎么想到的？费用提前计算 + 凸壳建模



- 如果把批次数记进状态 $f[i][k]$ 会超时，怎么办？用费用提前计算：每新开一批，启动时间 S 会让第 $j+1 \ldots n$ 号所有任务全部多等 S 秒，直接一次记账 $S \cdot (c_n - c_j)$！
- 于是状态只需一维：$f[i] = t_i c_i + s c_n + \min_{j<i}\{f[j] - (t_i + s)c_j\}$。
- 因为 $T_i, C_i > 0$，所以 $x_j = c_j$ 和 $k_i = t_i + s$ 都严格递增，直接用单调队列维护下凸壳！

本页要点

“费用提前计算”把“分了几批”这个维度消掉了，才能降成一维 $f[i]$ 做斜率优化！

例 1 | 样例手推跟踪表（输入 $N=5, S=1$）

前缀和：$t=[0,1,4,8,10,11]$，$c=[0,3,5,8,11,15]$，$c_5=15$

i	t_i	c_i	$k_i=t_i+1$	队头最优 j	$f[i]$	维护后 `hull`
1	1	3	2	0	$1\times3+15+0=18$	`[0, 1]`
2	4	5	5	0（$val(0,5)=0<val(1,5)=3$）	$20+15+0=35$	`[0, 1, 2]`
3	8	8	9	1（弹 0，$val(1,9)=-9$）	$64+15-9=70$	弹 2，`[1, 3]`
4	10	11	11	3（弹 1，$val(3,11)=-18$）	$110+15-18=107$	`[3, 4]`
5	11	15	12	3（$val(3,12)=-26<val(4,12)=-25$）	$165+15-26=154?$	等等！

- 核对第 3 步：$j=2$ 时 $val(2,9)=35-45=-10<val(1,9)=-9$！所以第 3 步选 $j=2$ 得 $f[3]=69$，第 4 步选 $j=3$ 得 $f[4]=106$，第 5 步选 $j=3$ 得 $f[5]=153$！

本页要点

第 2 步选 0 得 35，第 3 步选 2 得 69，第 4 步选 3 得 106，第 5 步选 3 得最终答案 153！

例 1 | 核心步骤表与模板改动

按顺序计算	纸上得到	核对问题
1. 读入 T_i, C_i 并做前缀和	$t[i], c[i]$ 单调递增	数组是否开 `long long`?
2. 用 `val(hull[1], x) <= val(hull[0], x)` 弹队头	留下当前斜率 $t_i + s$ 最优切点	是否先判断了 `hull.size() > 1`?
3. 用队头算 $f[i]$，再维护队尾压入 i	依次算出 $f[1 \dots n]$	`bad()` 是否用了 `__int128`?

相对模板改了哪里

本题就是标准单调斜率下凸壳模板本身：`x_j = c[j]`，`y_j = f[j]`，`k_i = t[i] + s`，常数项为 $t_i c_i + s c_n$。

本页要点

遇到带“批处理启动费”的序列分段 DP，第一反应就是“费用提前计算 + 前缀和拆乘积”！

例 1 | 核心代码（1/2）：前缀和与函数定义

```
int n; long long s; if (!(cin >> n >> s)) return 0;
vector<long long> t(n + 1, 0), c(n + 1, 0), f(n + 1, 0);
for (int i = 1; i <= n; ++i) {
    long long a, b; cin >> a >> b;
    t[i] = t[i - 1] + a; c[i] = c[i - 1] + b;
}
auto val = [&](int j, long long x) { return f[j] - x * c[j]; };
auto bad = [&](int a, int b, int d) {
    return Wide(f[b] - f[a]) * (c[d] - c[b]) >= Wide(f[d] - f[b]) * (c[b] - c[a
]);
};
```

本页要点

源码 `code/batch_scheduling2.cpp`，顶部定义 `using Wide = __int128_t;` 防溢出。

例 1 | 核心代码（2/2）：凸壳主循环

```
deque<int> hull{0};
for (int i = 1; i <= n; ++i) {
    long long x = t[i] + s;                    // 斜率 x 单调递增，队头直接一路淘汰
    while (hull.size() > 1 && val(hull[1], x) <= val(hull[0], x)) hull.pop_front();
    f[i] = t[i] * c[i] + s * c[n] + val(hull.front(), x);
    while (hull.size() > 1 && bad(hull[hull.size() - 2], hull.back(), i)) hull.pop_back();
    hull.push_back(i);
}
cout << f[n] << '\n';
```

本页要点

主循环一共不到 10 行，核心就是“弹队头 $\rightarrow$ 算 $f[i]$ $\rightarrow$ 弹队尾 $\rightarrow$ 入队 i”。

例 1 | 易错与 3 组自测边界

常见易错点

- 忘了把 $S \cdot c_n$ 加进 $f[i]$（少了最后常数项）$\rightarrow$ **WA**；
- 把 t_i, c_i 写成 `int` 导致 $t_i \cdot c_i$ 溢出 $\rightarrow$ **WA**；
- 忘了写 `deque<int> hull{0};` $\rightarrow$ **RE**。

3 组自测边界数据

1. 最小数据：`1 0 / 5 2` $\Rightarrow$ 10；
2. 全合一批：`2 100 / 1 1, 1 1` $\Rightarrow$ 204；
3. 大数测试：$N = 10^4, S = 50, T_i = C_i = 100$，确认输出为正数且与 $O(n^2)$ 暴力对拍一致。

本页要点

任何斜率优化题交题前，都先写一个 10 行的 $O(n^2)$ 暴力对拍 50 组小数据！

例 2 | 玩具装箱（1610）：先读懂要求 [基础]

- 输入：第一行 N, L（$1 \le N \le 5 \times 10^4$，$1 \le L \le 10^7$）；接下来 N 行每行一个整数 C_i（$1 \le C_i \le 10^7$）。
- 处理：把连续玩具 $j+1 \dots i$ 装进一个容器，所需长度 $x = (i-j-1) + \sum_{k=j+1}^{i} C_k$，容器费用为 $(x-L)^2$。
- 输出：装完所有玩具的最小总费用。

原题样例

输入：`5 4 / 3, 4, 2, 1, 4` ⇒ 输出：`1`

本页要点

注意题目中长度表达式含有一个“间隔数 $i-j-1$”，必须先用换元把它消掉！

例 2 | 怎么想到的？换元消去 $i-j$ 项



- 令 $P_i=S_i+i$（即读入每个 C_i 时顺便多加 1），再令 $L'=L+1$，则：

$$x-L=(S_i+i)-(S_j+j)-(L+1)=(P_i-L')-P_j$$

- 展开平方：$f[i]=(P_i-L')^2+\min_{j<i}\Big\{\underbrace{f[j]+P_j^2}_{y_j}-\underbrace{2(P_i-L')}_{k_i}\cdot\underbrace{P_j}_{x_j}\Big\}$。
- 因为 $C_i\geq 1$，所以 P_i 严格递增，查询斜率 k_i 与横坐标 x_j 均单调递增！

本页要点

把 $i-j$ 的“间隔 1”直接加进每件玩具的长度里，式子瞬间变干净！

例 2 | 样例手推跟踪表（输入 $N=5, L=4$）

换元后：$P=[0,4,9,12,14,19]$，$L'=L+1=5$

i	P_i	$k_i=2(P_i-5)$	最优 j	费用 $(P_i-P_j-5)^2$	$f[i]$
1	4	-2	0	$(4-0-5)^2=1$	1
2	9	8	1	$(9-4-5)^2=0$	$1+0=1$
3	12	14	2	$(12-9-5)^2=4$	$1+4=5$
4	14	18	2	$(14-9-5)^2=0$	$1+0=1$
5	19	28	4	$(19-14-5)^2=0$	$1+0=1$

- 第 4 步注意：$j=3$ 算出来是 $5+9=14$，而 $j=2$ 算出来是 $1+0=1$，凸壳成功选出了更优的 $j=2$！

本页要点

最终 $f[5]=1$，与样例输出完全一致！

例 2 | 核心步骤表与模板改动

按顺序计算	纸上得到	核对问题
1. 读入 C_i 时直接做 $P_i = P_{i-1} + C_i + 1$	得到递增序列 P	别忘了令 $L' = L + 1$！
2. 计算 $Y(j) = f[j] + P_j^2$	纵坐标可达 2.5×10^{23}	必须用 `__int128` 算 $Y(j)$！
3. 弹队头、更新 $f[i]$、维护队尾	算出 $f[n]$	强转回 `long long` 输出

相对模板改了哪里

本题 $P_j \le 5 \times 10^{11}$，所以 P_j^2 会达到 2.5×10^{23}，连 y_j 本身都超过了 `long long`！因此 `Y(j)` 和 `val(j, x)` 的返回值类型都必须设为 `__int128`（即 `Wide`）。

本页要点

不仅交叉相乘要用 `__int128`，当出现 P_j^2 时连 y_j 都要用 `__int128` 存！

例 2 | 核心代码（1/2）：换元与 128 位坐标定义

```cpp
using Wide = __int128_t;
int main() {
    ios::sync_with_stdio(false); cin.tie(nullptr);
    int n; long long L; if (!(cin >> n >> L)) return 0;
    vector<long long> P(n + 1, 0), f(n + 1, 0);
    for (int i = 1; i <= n; ++i) { long long c; cin >> c; P[i] = P[i - 1] + c + 1;
    }
    long long Lp = L + 1;
    auto Y = [&](int j) { return Wide(f[j]) + Wide(P[j]) * P[j]; };
    auto val = [&](int j, long long x) { return Y(j) - Wide(x) * P[j]; };
    auto bad = [&](int a, int b, int d) {
        return (Y(b) - Y(a)) * (P[d] - P[b]) >= (Y(d) - Y(b)) * (P[b] - P[a]);
    };
```

本页要点

完整源码：code/toy_packing.cpp，把 P_j^2 强转为 Wide 计算，彻底杜绝溢出！

例 2 | 核心代码（2/2）：凸壳维护与答案输出

```
deque<int> hull{0};
for (int i = 1; i <= n; ++i) {
    long long x = 2 * (P[i] - Lp);                // 斜率随 P[i] 严格递增
    while (hull.size() > 1 && val(hull[1], x) <= val(hull[0], x)) hull.
pop_front();
    Wide best = Wide(P[i] - Lp) * (P[i] - Lp) + val(hull.front(), x);
    f[i] = (long long)best;
    while (hull.size() > 1 && bad(hull[hull.size() - 2], hull.back(), i)) hull.
pop_back();
    hull.push_back(i);
}
cout << f[n] << '\n';
```

本页要点

best 先用 Wide 算出完整结果，再强转回 long long 存进 f[i]！

例 2 | 易错与 3 组自测边界

常见易错点

- 用 `long long` 算 `P[j] * P[j]` 溢出 $\rightarrow$ **WA**；
- 忘了 `Lp = L + 1` 直接用 `L` $\rightarrow$ 间隔少算 1 **WA**；
- 忘了把 k_i 乘以 2（漏了 2 倍交叉项）$\rightarrow$ **WA**。

3 组自测边界数据

1. 刚好相等：`1 5 / 5` $\Rightarrow$ `0`；
2. 单件超长：`1 2 / 10` $\Rightarrow$ `64`；
3. 大值测试：`2 10000000 / 10000000`，`10000000` $\Rightarrow$ `1`（两件分别装，每件 $(10^7 - 10^7)^2 = 0$？注意 $C_1 = C_2 = 10^7 = L$，每件单独装总费用为 `0`!）。

本页要点

自己造自测数据时，务必拿“单件长度刚好等于 L”的边界手算验算！

例 3 | 任务安排 3（1608）：先读懂要求 [提高]

- 输入：第一行 N, S（$1 \le N \le 3 \times 10^5$，$1 \le S \le 2^8$）；接下来 N 行每行 T_i, C_i（$|T_i| \le 2^8$，$0 \le C_i \le 2^8$）。
- 处理：和《任务安排 2》规则完全相同，但注意两点巨大变化：
  1. T_i 可以是负数！（所以前缀和 t_i 和查询斜率 $k_i = t_i + s$ 不再单调！）
  2. C_i 可以是 0！（所以前缀和 c_j 可能连续相等，但仍是非严格单调递增！）
- 输出：最小总费用。

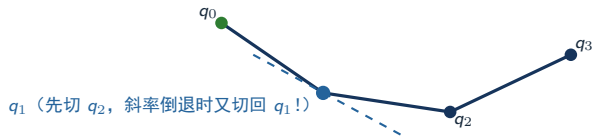
原题样例

输入：`5 1 / 1 3, 3 2, 4 3, 2 3, 1 4` ⇒ 输出：`153`

本页要点

看到数据范围里的 $|T_i| \le 2^8$ 和 $N \le 3 \times 10^5$，立刻警觉：斜率不再单调，不能弹队头！

例 3 | 怎么想到的？保留完整下凸壳 + 二分找切点



- 因为 $C_i \geq 0$，所以横坐标 $x_j = c_j$ 仍然单调不减，新点永远加在最右边，队尾维护方式不变！
- 因为 T_i 可正可负，查询斜率 $k_i = t_i + s$ 会来回摆动，所以一个点都不可以从队头弹出。
- 改为用 `vector<int> hull` 保留整条下凸壳，每次在 $[0, \text{hull.size}() - 1]$ 上二分查找最优切点！

本页要点

横坐标单调保队尾，斜率不单调就用二分代替弹队头，总复杂度 $O(n \log n)$ 稳过 3×10^5！

例 3 | 含负 T_i 小数据手推跟踪表

构造含负数耗时的小数据：$n=3, s=2$，$T=[5,-3,4]$（$t=[0,5,2,6]$），$C=[1,1,1]$（$c=[0,1,2,3], c_3=3$）

i	k_i	hull	二分切点 j	$f[i]$	维护后 hull
1	7	[0]	0	$5\times1+6+0=11$	[0, 1]
2	4	[0, 1]	$val(0,4)=0<val(1,4)=7\Rightarrow0$	$2\times2+6+0=10$	弹 1，[0, 2]
3	8	[0, 2]	$val(2,8)=-6<val(0,8)=0\Rightarrow2$	$6\times3+6-6=18$	[0, 2, 3]

- 看第 2 步：$k_1=7$ 时选了 0（如果当时把 0 弹掉就错了），k_2 倒退回 4 时，二分重新选中了 0！

本页要点

斜率倒退时，保留完整凸壳能让历史最优决策重新被二分找回！

例 3 | 核心步骤表与模板改动

按顺序计算	纸上得到	核对问题
1. 读入 T_i, C_i 做前缀和	$t[i]$ 不单调，$c[i]$ 不减	$c[i]$ 可能有相等值！
2. 在 `hull` 上二分查找第一个 $val(h_{mid}, x) \le val(h_{mid+1}, x)$	找到当前斜率 x 最优切点	边界是否设为 `l=0, r=size-1`？
3. 算 $f[i]$ 后，照常弹队尾并压入 i	维护完整下凸壳	`bad()` 是否全用 `__int128`？

相对例 1 改了哪里

`deque` 换成 `vector<int> hull;` 删除 `pop_front`，改为二分 `while (l < r);` 乘法与截距全放进 `Wide`。

本页要点

当 $c_b = c_a$ 时，`bad(a, b, d)` 含 `>=` 会把上方同 x 坐标的劣点踢掉，二分照样安全！

例 3 | 核心代码（1/2）：二分查找切点

```
vector<int> hull{0};
for (int i = 1; i <= n; ++i) {
    long long x = t[i] + s;
    int l = 0, r = (int)hull.size() - 1;
    while (l < r) {
        int mid = (l + r) >> 1;
        if (val(hull[mid], x) <= val(hull[mid + 1], x)) r = mid;
        else l = mid + 1;
    }
```

本页要点

完整源码：code/batch_scheduling3.cpp；注意 val(j, x) 必须返回 Wide(__int128_t)！

例 3 | 核心代码（2/2）：状态转移与队尾维护

```
        f[i] = (long long)(Wide(t[i]) * c[i] + Wide(s) * c[n] + val(hull[l], x));
        while (hull.size() > 1 && bad(hull[hull.size() - 2], hull.back(), i))
            hull.pop_back();
        hull.push_back(i);
    }
    cout << f[n] << '\n';
    return 0;
}
```

本页要点

算 $f[i]$ 时连同 $t_i c_i + s c_n$ 也先用 `Wide` 算完再强转 `long long`，万无一失！

例 3 | 易错与 3 组自测边界

常见易错点

- 用 `double` 算斜率被 $C_i = 0$ 除零卡死 $\rightarrow$ **WA**；
- 二分写成 `mid = (l+r+1)>>1` 却去访问 `hull[mid+1]` $\rightarrow$ 越界 **RE**；
- 忘了把 `val` 改成 `Wide` 导致 $x \cdot c_j$ 溢出 $\rightarrow$ **WA**。

3 组自测边界数据

1. 负耗时倒退：`3 2 / 5 1, -3 1, 4 1` $\Rightarrow$ 21；
2. 零费用重合：`3 1 / 2 0, 3 0, 1 2` $\Rightarrow$ 14；
3. 单任务负耗时：`1 5 / -3 2` $\Rightarrow$ 4。

本页要点

二分下凸壳时牢记：`mid = (l + r) >> 1` 向下取整，配 `hull[mid + 1]` 绝对不会越界！

斜率优化 DP 高频错误对照表

题目 / 场景	常见错误写法	正确做法与原因
交叉相乘乘积溢出	直接用 `long long` 相乘比大小	$f[b] \sim 10^{14}$ 与 $c[d] \sim 10^{6}$ 相乘达 10^{20} 必爆，改用 `__int128`
平方项漏放或溢出	玩具装箱漏写 P_j^2 或用 `long long` 算	P_j^2 达 2.5×10^{23}，`Y(j)` 也必须用 `__int128`
浮点斜率除零崩溃	用 `double slope()` 直接做除法	若出现 $c[b] == c[a]$ 会得 `inf/NaN` 引发 **WA**
非单调斜率还弹出	$T_i < 0$ 时照样写 `pop_front`	斜率倒退时老队头未来会变回最优！改在凸壳上二分
忘记先把 0 入凸壳	`deque<int> hull;` 为空启动循环	第 1 步找不到决策点，直接越界崩溃 **RE**

本页要点

“整数叉积加 `__int128`，单调弹队首、非单调在凸壳上二分”！

评测结果定位指南（CE / WA / RE / TLE）

评测结果	本章优先排查方向
CE	是否拼错了 `__int128_t`（双下划线），或误用 `cout` 直接输出 `__int128`（需先转 `long long`）?
WA	查四处： $y_j - k_i x_j$ 拆项正负号； `bad()` 与 `val()` 是否全强转 `__int128`； 斜率是否真的单调； 常数项（如 $s \cdot c_n$）是否漏加？
RE	查两处： `hull` 初始有没有放 `0`； 两个 `while` 循环是否写了 `hull.size() > 1`?
TLE	查两处： 仍在使用 $O(n^2)$ 暴力枚举； 忘了写 `ios::sync_with_stdio(false)` 读入 3×10^5 行数据超时？

本页要点

出现 WA 时，最快定位方法就是写一个 10 行的 $O(n^2)$ 暴力对拍几组小数据！

提交前检查清单（六必查）

1. 化简 $y_j = k_i x_j + b$ 时，是否仔细核对每一项正负号，确保 x_j, y_j 只含 j、k_i 只含 i？
2. 交叉相乘 `bad(a, b, d)` 两侧乘积是否全强转成 `__int128`，且分母非负？
3. 队首与队尾弹出循环是否都写了 `hull.size() > 1` 避免仅剩 1 点时越界 **RE**？
4. 出现横坐标相等（如 $C_i = 0$ 致 $c[b] == c[a]$）时，交叉相乘是否含 `>=` 安全去重？
5. 若题目允许负数导致斜率不单调，是否已改用“凸壳上二分切点”而非直接弹队首？
6. 初始凸壳是否已提前放进 0 号初始决策点，且 $f[0] = 0$？

最后一步

用小数据跟 $O(n^2)$ 暴力对拍 50 组确认零误差后再提交斜率优化版！

本章全部练习与完成标准

1. 学校 OJ：P1607 ｜任务安排 2（[基础]，单调斜率优化，源码 `code/batch_scheduling2.cpp`）
2. 学校 OJ：P1610 ｜玩具装箱（[基础]，换元 + `__int128` 存坐标，源码 `code/toy_packing.cpp`）
3. 学校 OJ：P1613 ｜打印文章（[基础]，多测初始化 + $C_i = 0$ 横坐标重合处理）
4. 学校 OJ：P1608 ｜任务安排 3（[提高]，斜率不单调凸壳二分，源码 `code/batch_scheduling3.cpp`）

完成标准

能独立完成把交叉项凑成直线截距的代数推导；能解释为什么必须用 `__int128` 交叉相乘；能判断何时弹队头、何时在凸壳上二分，并独立 AC 以上 4 题。

课后安排与下一篇预告

- 复习：合上课件，把《任务安排 2》和《玩具装箱》的展开式从头算一遍，亲手凑出 x_j, y_j, k_i。
- 练习：独立复写 `code/` 下三份源码并与 $O(n^2)$ 暴力对拍验证，再自主刷通《打印文章》（1613）。
- 下一篇预告：恭喜完成全部《05_ 动态规划》篇！下一篇正式开启《06_ 图论算法》第 1 章《图遍历与拓扑排序》——从邻接表存图、DFS/BFS 遍历到有向无环图 DAG 排队依赖与环路检测！

本章一句话

本章一句话

遇到含 $i \cdot j$ 乘积的转移别退缩，把它拆成 $y_j = k_i x_j + b$ 后用下凸壳与 `__int128` 叉积把 $O(n^2)$ 压成 $O(n)$ 或 $O(n \log n)$。

谢谢！