贪心策略与邻项交换

局部最优通往全局最优：用“交换不等式”彻底消除靠猜写题

HGNU-ACM 实验室算法组

承接第 4 章《二分答案与三分法》，掌握无需猜答案的局部最优证明与经典贪心模型





本章目录（1/2）

局部最优何以保证全局最优

邻项交换论证法：把“排序规则”算出来

区间调度与按右端点排序

前三节建立贪心底气：局部最优何时保全局、邻项交换不等式（含《国王游戏》）、区间按右端点排序。

本章目录（2/2）

哈夫曼合并与栈式/纸牌贪心

四大贪心经典真题精讲

常见错误、检查清单与练习安排

后三节掌握哈夫曼合并与栈式贪心，精讲《排队接水》《活动选择》《均分纸牌》《删数问题》与收尾清单。

生活小例子：什么时候“眼下最划算”是对的，什么时候翻车？

第 4 章我们靠“猜一个答案再去校验”求最值；而如果每一步不用猜，直接挑眼下看起来最好的那一步就能一直走到最优终点，这就是贪心策略。

成功：人民币找零 $\{10, 5, 1\}$ 凑 18

每次拿能放下的最大面额：
$18 \to$ 先拿 10（剩 8）$\to$ 再拿 5（剩 3）$\to$ 拿 3 个 1。一共 5 张，绝对最少！

翻车：奇异硬币 $\{11, 5, 1\}$ 凑 15

“每次拿尽量大”：先拿 11（剩 4），只能拿 4 个 1，用了 5 枚；
但最优解其实是 $5 + 5 + 5 = 15$，只要 3 枚！

本页要点

同样是“每次挑最大”，换一组面额就会 WA！写贪心前必须先想明白“为什么不会后悔”。

为什么暴力枚举所有选法会 TLE？

在组合或排布问题里，如果不做贪心选择而是把所有可能方案全部试一遍：

- 选或不选类问题（如选尽量多活动）：n 个物品各有选/不选两种可能，共 2^n 种组合。当 $n = 100$ 时，$2^{100} \approx 1.26 \times 10^{30}$，直接 **TLE**！
- 排列顺序类问题（如 n 个人排队接水）：n 个人有 $n!$ 种排队方式。就算 $n = 15$ 也有 1.3×10^{12} 种，必须用 $O(n \log n)$ 排序替代暴力。

规模 n	暴力枚举 2^n 或 $n!$	贪心排序 $O(n \log n)$	结论
10	1024 / 3.6×10^6	≈ 33 次	暴力勉强，贪心极快
100	2^{100} 宇宙爆炸	≈ 664 次	暴力必 **TLE**，贪心毫秒过
10^5	无法计数	$\approx 1.7 \times 10^6$ 次	只有贪心/DP 能过

本页要点

贪心能把指数/阶乘级的暴力搜索瞬间降成线性或 $O(n \log n)$，前提是策略本身必须对。

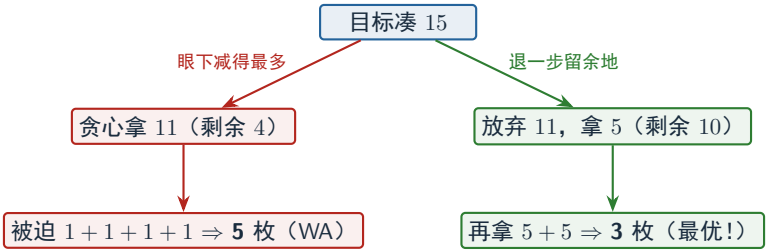
三个新术语大白话

- 贪心选择性质：不需要把所有未来分支都算完才知道哪步好——当前这一步选出局部最优后，一定能顺着它继续凑出全局最优解，永远不会因为“现在贪了小便宜”而后面无路可走。
- 最优子结构：走完第一步贪心选择后，剩下的半截问题和原问题长得一模一样（只是规模变小了），且大问题的最优解里包含着剩下小问题的最优解。
- 反例证伪法：检验一个拍脑袋想出的贪心规则有没有漏洞，最快的方法不是写代码交 OJ，而是自己造 $2 \sim 3$ 个只含 3 个元素的极端小例子，看能不能找到比它更优的解。

本页要点

“找反例”只要 30 秒就能救你一次 WA；找不到反例再动手证明为什么一定对。

TikZ 图解：硬币 $\{1, 5, 11\}$ 凑 15 为何贪心被“锁死”



本页要点

硬币问题因为面额之间不是整倍数关系，拿了 11 会破坏 5 的整除组合，导致不满足贪心选择性质。

手算跟踪表：成功案例 $\{10,5,1\}$ 凑 18 vs 翻车案例 $\{11,5,1\}$ 凑 15

统一规则：每次从大到小选第一个 $\le$ *rem* 的面额，直到 $rem == 0$ 退出循环：

案例	轮次	$rem > 0$?	贪心选币	剩余 *rem*（累计枚数）
成功 $\{10,5,1\}$	1	真（$18 > 0$）	拿 10	8（已用 1 枚）
成功 $\{10,5,1\}$	2	真（$8 > 0$）	拿 5	3（已用 2 枚）
成功 $\{10,5,1\}$	$3 \sim 5$	真	拿 $1,1,1$	0（已用 **5** 枚，最优）
成功结束	6	假（$0 > 0$ 假）	—	退出，输出 5
翻车 $\{11,5,1\}$	1	真（$15 > 0$）	拿 11	4（已用 1 枚）
翻车 $\{11,5,1\}$	$2 \sim 5$	真	拿 $1,1,1,1$	0（已用 5 枚，非最优 3）
翻车结束	6	假（$0 > 0$ 假）	—	退出，错误输出 5

本页要点

算法同样会终止并给出合法凑法，但 $\{11,5,1\}$ 给出的并不是最少枚数！

正确性判别准则：什么时候敢写贪心，什么时候换 DP？

写题前用“替换法”自问两句话：

判别维度	可以放心用贪心	必须换用 DP 或二分/搜索
替换不更差？	把任意最优解的第一步换成我的贪心选择，剩余部分仍合法且代价不增	换完之后后续代价反而升高，或直接破坏约束（如硬币凑 15）
互相干扰吗？	只有“谁先谁后”或“右端点吞并”一种局部关系	选了当前项会彻底改变未来可用容量/状态（如 0-1 背包）

- 证明贪心的两大正规武器：(1) 邻项交换不等式（排序类）；(2) 替换/剪贴法（区间/选择类）。

本页要点

证明不了“换完绝不更差”，就老老实实造反例——贪心最怕“我觉得好像对”。

实验代码：面额贪心试探（验证反例只需 12 行）

```cpp
int c[3] = {11, 5, 1};                  // 面额降序排列
int rem, cnt = 0;
cin >> rem;
for (int i = 0; i < 3; ++i) {
    while (rem >= c[i]) {               // 只要当前最大面额装得下就拿
        rem -= c[i];
        ++cnt;
    }
}
cout << cnt << '\n';                    // rem == 0 退出后输出使用枚数
```

- 注意内层可以直接写成 `cnt += rem / c[i]; rem %= c[i];`，避免金额很大时循环太多次。

本页要点

面额大时别用 `while` 一枚枚减，直接用整除 `/` 和取模 `%` 一步到位。

上机验证与反事实追问

运行上面硬币程序（面额 {11，5，1}），测 3 组输入并与真实最少枚数对比：

- 正常命中：输入 16 ⇒ 程序输出 2（$11+5$，恰为最优）。
- 最小边界：输入 3 ⇒ 程序输出 3（$1+1+1$，最优）。
- 反例曝光：输入 15 ⇒ 程序输出 5（$11+1+1+1+1$），但真实最优是 **3**（$5+5+5$）！

反事实追问

如果把硬币面额改成 $\{50,10,5,1\}$（每一级都是前一级整数倍），任意凑 *rem* 贪心还会翻车吗？
答：不会！因为任意 5 枚 1 都能无损换成 1 枚 5，任意 2 枚 5 都能换成 1 枚 10……整倍数面额必然满足贪心选择性质！

贪心策略选择易错对照（WA）

易错场景	错误做法	正确做法与原因	评测结果
0-1 背包误用贪心	按性价比 v_i/w_i 从高到低装	物品不能切割，装满空余会浪费，必须用 DP	**WA**
非整倍硬币找零	盲目从大到小减	构造反例发现会卡死在局部，改用完全背包 DP	**WA**
未排好序就贪心	按输入原序扫一遍	大多数贪心依赖特定关键字序，先推导出排序条件	**WA**
忽略多解字典序	随便输出一种排序结果	按题面要求处理等值第二关键字或 `stable_sort`	**WA**

本页要点

贪心写完后几乎不会 TLE，出错绝大多数都是 WA——永远先怀疑“策略反例”。

生活小例子：两个人排队接水，谁排在前面？

具体例子：只有甲（需 5 分钟）和乙（需 2 分钟）打水，只有一个水龙头

- 若甲先乙后：甲等 0 分钟，乙等 5 分钟 $\Longrightarrow$ 两人总等待 $= 0 + 5 =$ **5** 分钟。
- 若乙先甲后：乙等 0 分钟，甲等 2 分钟 $\Longrightarrow$ 两人总等待 $= 0 + 2 =$ **2** 分钟。

只要让耗时短的乙站到前面，后面那个人的等待时间就立刻少等 3 分钟！

- 当有 n 个人排队时，如果存在一对相邻的人“前慢后快”，把他们俩交换一下，总等待时间一定会严格减少，而且完全不影响其他人的等待！

本页要点

不用猜整体怎么排：只要看“任意相邻两个人换一换划不划算”，就能推出全局唯一排序规则！

为什么暴力枚举全部排队顺序会 TLE？

如果有 n 个人排队（《排队接水》中 $n \le 1000$）：

- 暴力做法：用 `next_permutation` 枚举所有 $n!$ 种排队顺序，对每种顺序花 $O(n)$ 算总等待时间。
- 操作次数：哪怕只有 $n = 12$，排列数也有 $12! = 4.79 \times 10^8$，乘上计算早已超过 10^9 必 **TLE**；$n = 1000$ 更是天文数字。
- 邻项交换 + 排序：一旦推导出 $T_i \le T_j$ 应排在前面，直接调用 $O(n \log n)$ 的 `stable_sort`，当 $n = 1000$ 时只需约 10^4 次比较，瞬间 **AC**！

本页要点

邻项交换把 $O(n!)$ 的排列搜索直接简化为一次 $O(n \log n)$ 排序。

三个新术语大白话

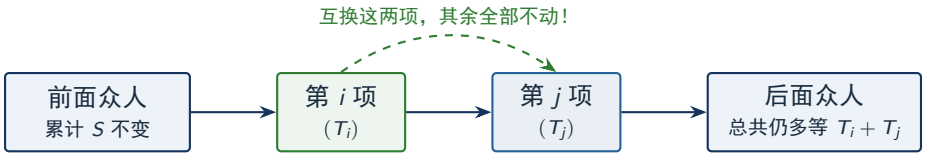
- 邻项交换（**Exchange Argument**）：假设有一个最优序列，随机挑出相邻的两个元素 i 和 j，算一算“i 在前 j 在后”与“j 在前 i 在后”的代价差，抵消掉前面和后面所有不变的部分，得到一个只含 i, j 自己参数的不等式。
- 无关项抵消：交换相邻两人时，排在他们前面的人已经接完了、排在他们后面的人看到前面总耗时仍然是 $T_i + T_j$，所以前后所有人的代价完全一样，直接从不等式两边约掉！
- 严格弱序比较器 `cmp(x, y)`：交给 C++ `sort` 的判断函数，回答“`x` 是否应该严格排在 `y` 前面”，相等时必须返回 `false`（不能写 `<=`）。

本页要点

只交换“相邻”两个元素，正是为了把中间无关人的影响彻底锁死成零！

TikZ 图解：邻项交换通用推导框架

设任意顺序中有一对相邻元素 i 在前、j 在后，前面众人的累计量为 S（或累乘为 P）：



- 因为交换满足加/乘法交换律，后面所有人感受到的“前面总耗时/总乘积”一模一样！
- 所以只需比较 i, j 这两项自身的局部代价，不等式干净利落！

本页要点

邻项交换三步曲：设相邻 (i, j) 互换 $\rightarrow$ 抵消前后无关项 $\rightarrow$ 化简出仅含 i, j 的排序不等式。

推导一：排队接水不等式与 4 人手算交换表

- 数学推导：设前面人已用时 S。
  - i 前 j 后时，i,j 总等待 $= S + (S + T_i) = 2S + T_i$；
  - j 前 i 后时，i,j 总等待 $= S + (S + T_j) = 2S + T_j$。
  - 原顺序更优 $\iff 2S + T_i \le 2S + T_j \iff \mathbf{T_i \le T_j}$（按耗时升序）！

用 4 人耗时 [5，2，8，3] 演示逐对交换到最优的跟踪过程（总等待每次必减）：

轮次	存在相邻 $T_i > T_j$?	当前序列	总等待时间	交换动作
1	真（$5 > 2$）	[5，2，8，3]	$0 + 5 + 7 + 15 = 27$	交换 5 和 2
2	真（$8 > 3$）	[2，5，8，3]	$0 + 2 + 7 + 15 = 24$	交换 8 和 3
3	真（$5 > 3$）	[2，5，3，8]	$0 + 2 + 7 + 10 = 19$	交换 5 和 3
4	假（无逆序）	[2，3，5，8]	$0 + 2 + 5 + 10 = \mathbf{17}$	全部有序，退出！

本页要点

每一次消除“前大后小”的相邻逆序，总等待严格下降；直到无逆序可换就是全局最优！

推导二：国王游戏（NOIP 2012）经典不等式推导

题目背景

国王左手 a_0、右手 b_0 排最前；后面 n 位大臣各有 (a_i, b_i)。第 k 位大臣拿到的金币是“前面所有人左手乘积 $\div b_k$”。求怎么排大臣，使获赏最多的大臣拿到最少？

- 设相邻两人 i, j 前面所有人的左手乘积为 $P > 0$（交换 i, j 不影响前后其他人）。
- i 前 j 后时两人最大奖赏 $M_1 = \max\left(\frac{P}{b_i},\ \frac{Pa_i}{b_j}\right)$；$j$ 前 i 后时 $M_2 = \max\left(\frac{P}{b_j},\ \frac{Pa_j}{b_i}\right)$。
- 同除 P 再同乘 b_ib_j，等价比较 $\max(b_j, a_ib_i)$ 与 $\max(b_i, a_jb_j)$。
- 因 $a_i, a_j \geq 1$ 有 $a_ib_i \geq b_i$、$a_jb_j \geq b_j$，故若 $a_ib_i \leq a_jb_j$，必有 $M_1 \leq M_2$！

本页要点

按 $a_i \times b_i$ 升序排列！若乘积超 `long long`，结合第 3 章《高精度运算》做高精度乘除！

邻项交换自定义排序模板（注意严格弱序）

```
struct Node {
    long long a, b;                        // 如国王游戏中每位大臣的左右手数字
};
bool cmp(const Node &x, const Node &y) {
    return x.a * x.b < y.a * y.b;          // 严格小于 <，千万不能写成 <= !
}
// 主函数里:
sort(v.begin(), v.end(), cmp);             // 若相等时要保留输入原序，改用 stable_sort
```

- 为什么乘法前先用 `long long`？若 a, b 是 `int`，$a \times b$ 可能先溢出再比较，结构体成员直接开 `long long` 最稳妥。
- 什么时候用 `stable_sort`？像《排队接水》要求“耗时相同按原始编号小在前”时，`stable_sort` 保证相等元素不打乱原有相对顺序。

本页要点

把推导出的不等式里的 $\leq$ 去掉等号写成 <，放进 cmp 就完成了全部核心逻辑！

上机验证与反事实追问

把上面 `Node` 与 `cmp` 用在 3 组小测试数据上（输入 n 对 (a_i, b_i) 输出排序后序列）：

- 普通逆序：(3，4)，(1，5)，(2，2)，乘积分别为 $12, 5, 4 \Rightarrow$ 排成 (2,2)，(1,5)，(3,4)。
- 最小边界 $n = 1$：(7，3) $\Rightarrow$ 不发生交换直接输出 (7，3)。
- 全同乘积：(2，6)，(3，4)，(4，3)（全为 12），若用 `stable_sort` 保持原序 1，2，3。

反事实追问

如果手滑把 `cmp` 写成了 `return x.a * x.b <= y.a * y.b;`，在遇到全相等测试数据时可能会发生什么？

答：违反了 C++ `std::sort` 的“严格弱序”约定（自己和自己比返回了 `true`），当数据量稍大触发内省排序快排分区时，指针可能越界引发 **RE**（段错误）！

邻项交换与排序易错对照（WA / RE）

易错点	错误写法	正确做法与原因	评测结果
比较器用 <=	`return x.w <= y.w;`	违反严格弱序会导致越界崩溃，必须写 <	**RE**
只凭直觉挑排序键	国王游戏只按 a_i 或 b_i 排序	必须通过邻项交换不等式推导出 `a[i]*b[i]`	**WA**
相等项顺序打乱	同耗时用普通 `sort`	要求保持原编号顺序时，用 `stable_sort` 或比编号	**WA**
比较乘积未转类型	`int` 相乘直接溢出	写 `1LL * x.a * x.b < 1LL * y.a * y.b`	**WA**
忽略传递性	推出比较规则有环 $A < B < C < A$	检验化简后的表达式能否写成独立单值函数 $f(i) < f(j)$	**WA**

本页要点

记住：一个靠谱的邻项交换排序关键字，化简后通常能写成 $f(i) < f(j)$ 的形式，天然满足传递性！

生活小例子：只有一个大礼堂，怎样接待最多场活动？

具体例子：今天有 4 个社团想借大礼堂，时间段分别是 $[0,10],\ [1,4],\ [4,7],\ [7,10]$

同一时间只能办一场活动，且前一场在 4 结束、后一场在 4 开始允许相邻接续。

- 如果选开始最早的 $[0,10]$，一场就把整天占满了，一共只能办 **1** 场！
- 如果选 $[1,4],[4,7],[7,10]$，虽然每个开始都比 0 晚，但每次早早散场，一共办了 **3** 场！

本页要点

选活动不是看谁抢跑（开始早），也不是看谁最短，而是看“谁结束得最早、尽早腾出礼堂”！

为什么暴力枚举所有子集一定会 TLE？

在《活动选择》（1323）中，活动数量 $n \le 1000$：

- 暴力做法：用 DFS 或二进制枚举选出全部 2^n 个活动子集，再检查每对区间是否冲突。
- 操作次数：当 $n = 1000$ 时，$2^{1000} \approx 10^{301}$，连宇宙原子的总数都装不下，必 **TLE**！
- 右端点排序贪心：先把 n 个区间按结束时间 end_i 从小到大排好序（$O(n \log n)$），再从左往右扫一遍（$O(n)$），$n = 1000$ 时仅需约 10^4 次操作，瞬间 **AC**！

本页要点

把 $O(2^n)$ 变成 $O(n \log n)$，秘诀就在一个正确的排序关键字：end_i！

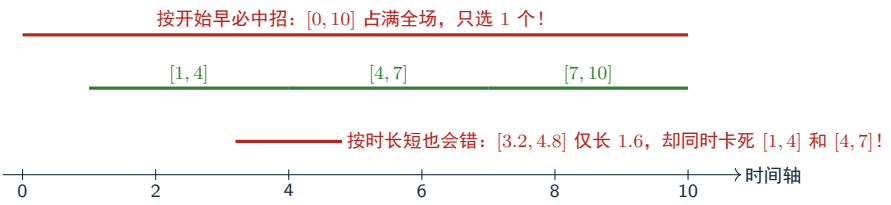
三个新术语大白话

- 区间相容（**Disjoint Intervals**）：两个时间段互相不打架，也就是后一个活动的开始时间 b_2 不早于前一个活动的结束时间 e_1。
- 右端点（结束时间 end_i）：区间 $[begin_i, end_i]$ 右边的边界点；选定一个活动后，接下来所有候选活动只能从 end_i 往右找。
- 闭区间衔接 `b >= last_end`：因为题面明确说明 $[1,4]$ 和 $[4,7]$ 可以相连安排，所以判断相容用大于等于 >=；如果某题要求端点重合也算冲突，才改成严格大于 >。

本页要点

先读清题面“端点重合算相容还是冲突”，再决定代码写 >= 还是 >。

TikZ 图解：反例否定“按开始早”和“按时长短”



本页要点

只有按“结束时间 end_i 从早到晚”排序，才能在每个局部做到“给后面留下尽量大的时间轴”。

手算跟踪表：按结束时间排序后筛选 5 个活动

排序后的活动为 $[1,4]$，$[3,5]$，$[0,6]$，$[5,7]$，$[8,11]$，初值 `last_end = -INF`，`cnt = 0`：

下标 i	当前 $[b,e]$	$b \geq$ `last_end`?	`cnt`	更新后 `last_end`	说明
0	$[1,4]$	真（$1 \geq -\infty$）	1	4	结束最早，必选
1	$[3,5]$	假（$3 < 4$）	1	4	开始太早与 $[1,4]$ 冲突，舍弃
2	$[0,6]$	假（$0 < 4$）	1	4	冲突，舍弃
3	$[5,7]$	真（$5 \geq 4$）	2	7	相容，选上并推进右端
4	$[8,11]$	真（$8 \geq 7$）	**3**	11	相容，选上
5	扫完数组	循环结束	3	11	输出最多选 **3** 个

本页要点

扫完最后一个区间后循环退出，全程只做了一次排序加一次线性扫描。

正确性证明：替换法 + 延伸到“整数区间选点”

- 为什么按结束最早选绝不会比任何最优解少？
  - 设某个真实最优方案按时间排好选了 k 个活动：$A_1, A_2, \ldots, A_k$。
  - 把所有活动里结束最早的那个记作 G_1，那么必然有 $end(G_1) \le end(A_1) \le begin(A_2)$。
  - 所以把最优方案的第 1 个活动直接替换成 G_1，得到新方案 $G_1, A_2, \ldots, A_k$：既完全不冲突，总数还是 k 个！
  - 接着把剩下的区间交给子问题递归归纳，证明全局最多就是贪心选出的个数！
- 一题通两题（**1324**《整数区间》）：给 n 个闭区间，选最少的整数点使每个区间至少含 1 个点。同样按右端点升序，若当前区间不含上一个选出的点 p（即 $a_i > p$），就贪心选该区间的最右端 $p = b_i$！

本页要点

“活动选择”和“整数区间选点”是对偶孪生模型：核心都是按右端点 end 从小到大排序！

区间调度标准模板代码（用 pair 省掉手写 cmp）

```
int n; cin >> n;
vector<pair<int, int>> act(n);          // {end, begin}，默认 sort 即按 end 升序！
for (int i = 0; i < n; ++i) cin >> act[i].second >> act[i].first;
sort(act.begin(), act.end());
int chosen = 0, last_end = INT_MIN;     // 极小负无穷作初值，防负数/零起点漏选
for (auto [e, b] : act) {
    if (b >= last_end) {                // 端点相接 [1,4] 与 [4,7] 也合法
        ++chosen; last_end = e;
    }
}
cout << chosen << '\n';
```

本页要点

把“结束时间”放在 pair.first、把“开始时间”放在 pair.second，连 cmp 都不用写！

上机验证与反事实追问

用上面模板测 3 组代表性数据：

- 全部重叠：`3 / 1 5 / 2 6 / 3 7` ⇒ 全部冲突，只能选最早结束的 `[1,5]`，输出 `1`。
- 全部互不冲突：`3 / 1 2 / 3 4 / 5 6` ⇒ 输出 `3`。
- 端点首尾相接：`2 / 1 4 / 4 7` ⇒ 输出 `2`。

反事实追问

1. 如果把 `last_end = INT_MIN;` 错设成 `0`，遇到含负时刻区间 `[-5, -1]` 时会怎样？
   答：`-5 >= 0` 为假，直接把本来能选的负数区间漏掉（WA）！
2. 如果题目要求端点重合 $[1,4]$ 和 $[4,7]$ 算冲突不能连着办，代码怎么改？
   答：只需把判断条件 `if (b >= last_end)` 改成严格大于 `if (b > last_end)` 即可！

区间贪心易错对照（WA）

易错点	错误写法	正确做法与原因	评测结果
排序关键字错	按 `begin` 或 `end - begin` 排	只能按右端点 `end` 升序（反例见图）	**WA**
`pair` 顺序写反	读入 `{b, e}` 却直接 `sort`	`pair` 默认先比 `first`，必须存 `{e, b}`	**WA**
相容条件缺等号	写了 `if (b > last_end)`	闭区间首尾相连合法时必须写 `>=`	**WA**
`last_end` 初值太高	设成 `0` 或 `1`	遇开始时间为 0 或负数时漏选，应设为 `INT_MIN`	**WA**

本页要点

看到区间类最值问题：第一反应画数轴举反例，然后试“按右端点排序”。

生活小例子：把果子两堆两堆合并成一堆，怎样总力气最小？

具体例子：地上有 3 堆果子重量 $[1,2,9]$，每次选两堆合起来，消耗体力等于两堆重量之和

- 笨做法：先合较重的 $2+9=11$，再合 $11+1=12 \implies$ 总体力 $=11+12=\mathbf{23}$。
- 聪明做法：先合最轻的 $1+2=3$，再合 $3+9=12 \implies$ 总体力 $=3+12=\mathbf{15}$！

- 为什么省了这么多？因为越早合并的堆，它的重量会在后面被反复累加很多次！把最轻的放最底层先合并，最重的放最后只算一次，总体力就最小！

本页要点

合并果子核心法则：每一轮都从当前所有堆里挑出“最轻的两堆”合并，然后把新堆放回去！

为什么暴力枚举合并树会 TLE，只排一次序也会 WA？

若有 $n = 10^4$ 堆果子：

- 暴力做法：枚举哪两堆先合、哪两堆后合，共有 $(n-1)!$ 种合并树（指数/卡特兰级爆炸），必 **TLE**！
- 常见错误贪心：只把原始数组 `sort` 一遍，然后从左到右顺着累加 ‘1+2 -> +4 -> +9‘。
  - 反例：$[3, 4, 5, 6]$。合完 $3 + 4 = 7$ 后，新堆 7 比剩下的 5 和 6 都重！下一步显然应该合 $5 + 6 = 11$，而不是继续合 $7 + 5$！
- 正确做法（小根堆 `priority_queue`）：每次用 $O(\log n)$ 弹出最小两堆，合并后再把新重量压回小根堆，总共只需 $O(n \log n)$，既快又对！

本页要点

新合并出来的大堆还会参与后续竞争，所以必须放回动态维护最小值的小根堆里！

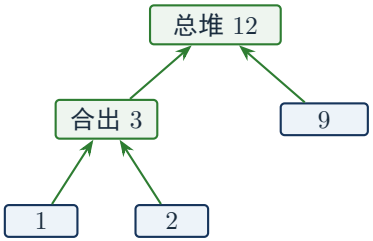
三个新术语大白话

- 哈夫曼树（**Huffman Tree**）：把每次“二合一”画成一棵二叉树，最先被合并的最轻果子落在最深层（经过次数最多），总代价等于所有中间合并节点之和（也等于每个果子重量 $\times$ 它被合并的深度）。
- 小根堆（**Min-Heap**）：C++ STL 里的 `priority_queue<long long, vector<long long>, greater<long long>>`，能在 $O(\log n)$ 内随时取出并删除当前最小的那个数，也能 $O(\log n)$ 插入新数。
- 单调栈贪心（字典序/数值最小）：从左到右扫数字串时，一旦发现栈顶高位数字比刚读到的新数字大，说明“留着旧高位会让整个数偏大”，只要还能删就果断弹出栈顶！

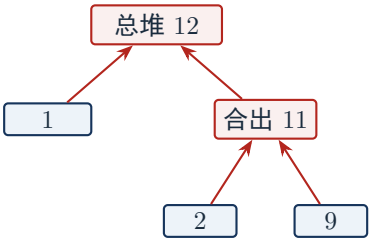
本页要点

哈夫曼靠“小根堆维护最小两堆”，删数靠“单调栈维护高位尽可能小”。

TikZ 图解：哈夫曼合并树对比 $[1, 2, 9]$



代价：$3 + 12 = \mathbf{15}$（$1, 2$ 乘深 2，9 乘深 1）



代价：$11 + 12 = \mathbf{23}$（9 被算了两遍！）

本页要点

叶子越深被重复加的次数越多，所以必须把最重的果子留到最浅层最后合！

手算跟踪表：用小根堆合并 4 堆果子 $[1, 2, 4, 9]$

每次循环判断堆大小是否 > 1；只要为真就弹出最小两堆相加，把和塞回堆中并累加体力：

轮次	`pq.size() > 1`?	弹出最小两堆	合并新堆	累计总代价	操作后堆内元素（升序看）
初始	—	—	—	0	$\{1, 2, 4, 9\}$
1	真（$4 > 1$）	1, 2	3	3	$\{3, 4, 9\}$
2	真（$3 > 1$）	3, 4	7	$3 + 7 = 10$	$\{7, 9\}$
3	真（$2 > 1$）	7, 9	16	$10 + 16 = \mathbf{26}$	$\{16\}$
4	假（$1 > 1$ 假）	—	—	退出循环	只剩 1 堆，输出 **26**

本页要点

当堆里只剩 1 个元素时 `pq.size() > 1` 为假，循环结束；注意 $n = 1$ 时本来就只有一堆，代价是 0！

正确性论证：最深层叶子替换法

- 为什么全局最优合并树的最底层，一定能放最小的两堆 x_1, x_2？
  1. 在任一最优合并树里，挑出深度最深（被累加次数 d 最多）的那对兄弟节点 a, b。
  2. 假设最小的两堆 x_1, x_2 深度分别为 $d_1, d_2 \le d$。
  3. 如果把 x_1 和 a 交换位置，总代价的变化量是：

$$\Delta = (a - x_1)(d_1 - d) \le 0 \quad (\text{因为} a \ge x_1 \text{ 且} d_1 \le d).$$

  4. 所以把最小的两堆放到最深层合并成一棵新节点 $x_1 + x_2$，绝对不会让总代价变大！接着问题规模严格减 1，归纳即证。

本页要点

“重者浅、轻者深”的交换不等式证明了哈夫曼贪心的绝对正确性。

哈夫曼合并小根堆模板代码

```
int n; cin >> n;
priority_queue<long long, vector<long long>, greater<long long>> pq;
for (int i = 0; i < n; ++i) {
    long long x; cin >> x; pq.push(x); // 所有初始堆入小根堆
}
long long ans = 0;
while (pq.size() > 1) {                  // 至少还有两堆才需要合并
    long long a = pq.top(); pq.pop();    // 取出第 1 小
    long long b = pq.top(); pq.pop();    // 取出第 2 小
    ans += a + b;                        // 累加本次合并体力
    pq.push(a + b);                      // 新堆必须放回小根堆继续竞争!
}
cout << ans << '\n';
```

本页要点

记住小根堆的三参数固定写法：`priority_queue<T, vector<T>, greater<T>>`。

上机验证与反事实追问

运行上面哈夫曼代码，验证以下 3 组输入：

- 经典 **3** 堆：`3 / 1 2 9` $\Rightarrow$ 输出 `15`。
- 最小边界 $n = 1$：`1 / 100` $\Rightarrow$ 不进入 `while` 循环，直接输出 `0`（不用搬运）。
- 需重新插回排序的 **4** 堆：`4 / 3 4 5 6` $\Rightarrow$ 先合 $3+4=7$，再合 $5+6=11$，再合 $7+11=18$，输出 `36`。

反事实追问

如果图省事不用 `priority_queue`，只把 `[3, 4, 5, 6]` 排序后从左往右前缀累加，结果会是多少？

答：会算出 $(3+4)+(7+5)+(12+6)=7+12+18=37>36$，直接 **WA**！因为 7 没有被放到 $5,6$ 后面！

哈夫曼与栈式贪心易错对照（WA / RE）

易错点	错误写法	正确做法与原因	评测结果
默认建成了大根堆	`priority_queue<long long> pq;`	默认每次弹出最大值，必须加 `greater<>` 改小根堆	**WA**
只排一次序顺加	排序后直接依次叠加前缀	合并产生的新堆可能大于未合并堆，必须重入堆	**WA**
循环条件误写非空	`while (!pq.empty())` 连 pop 两次	最后只剩 1 堆时第二次 pop 崩溃，写 `pq.size() > 1`	**RE**
总代价变量溢出	`int ans = 0;`	多次累加可达 10^{10} 以上，统一开 `long long`	**WA**

本页要点

使用优先队列连取两个元素前，循环条件务必写 `pq.size() > 1`，防止空堆弹爆！

例 1 | 排队接水（一本通 1319）[入门]：先读懂要求

大白话题意

有 n 个人在同一个水龙头前排队接水，第 i 个人接水需要 T_i 个单位时间。
前一个人接水时后面所有人都得等着；本人接水的时间不算在自己的等待时间里。
请安排一种排队顺序，使所有人的平均等待时间最小；若多人耗时相同，按原始编号从小到大排。

- 输入：第一行 $n \le 1000$；第二行 n 个整数 $T_i \le 10000$。
- 处理：记录 $\{T_i,\ i\}$，按 T_i 升序稳定排序后累加等待时间。
- 输出：第一行输出 $1 \sim n$ 的最优编号排列；第二行输出最小平均等待时间（保留 2 位小数）。
- 原题样例：`n=10`，耗时 `56 12 1 99 1000 234 33 55 99 812` $\Rightarrow$ `3 2 7 8 1 4 9 6 10 5` 与 `291.90`。

本页要点

先别急着算平均值：读清“耗时相同时按原编号小在前”，选 `stable_sort` 一步搞定！

例 1 | 排队接水：怎么想到的与建模图解

- 由邻项交换得出：任意相邻两人 i, j 互换只影响他俩，总等待分别是 $2S + T_i$ 与 $2S + T_j$，因此按 T_i 从小到大排序必得全局最小总等待！
- 注意编号处理：样例里第 4 号与第 9 号耗时都是 99，输出顺序必须是 `4 9`，不能颠倒成 `9 4`。



本页要点

排第 k 位的人只需等前面 $k - 1$ 个人接完水，所以是“先加 `clock_sum` 到 `waited`，再把当前 T 加进 `clock_sum`”！

例 1 | 排队接水：样例前 5 人与全序列核对手推表

对样例 `56 12 1 99 1000 234 33 55 99 812` 排序后编号依次为 `3,2,7,8,1,4,9,6,10,5`：

位置	人员编号	耗时 T	本人等待（累计时钟 `clock_sum`）	累计等待和 `waited`
1	3	1	0	0
2	2	12	$0+1=1$	1
3	7	33	$1+12=13$	14
4	8	55	$13+33=46$	60
5	1	56	$46+55=101$	161
$6\sim10$	$4,9,6,10,5$	$99,99,234,812,1000$	$157,256,355,589,1401$	**2919**

本页要点

全部 10 人总等待和 2919，除以 10 得 291.90，与样例输出分毫不差！

例 1 | 排队接水：核心步骤与相对模板的变化

按顺序计算	纸上得到	核对问题
1. 绑定编号与耗时	`pair{T_i, i+1}`	原始编号从 1 开始，别错写 0
2. 稳定升序排序	`stable_sort` 按 `first` 升序	同耗时保持原编号顺序吗？
3. 累加等待总和	先 `waited += clock_sum`，再加 `first`	最后一人的接水耗时有没有误算进等待？

相对模板改了哪里

在邻项交换排序模板基础上，给每个元素绑定原始编号 `second = i + 1`，用 `stable_sort` 仅比较 `first`（或显式比较第二关键字编号），并用 `long long waited` 累加前缀和。

本页要点

总等待和最大约 $\frac{1000^2}{2} \times 10^4 = 5 \times 10^9$，超出 `int` 上限，必须用 `long long`！

例 1 | 排队接水：完整代码（1/2，code/water_queue.cpp）

```
int n;
if (!(cin >> n)) return 0;
vector<pair<int, int>> a(n);          // {接水时间, 原始编号}
for (int i = 0; i < n; ++i) {
    cin >> a[i].first;
    a[i].second = i + 1;
}
stable_sort(a.begin(), a.end(), [](const auto &x, const auto &y) {
    return x.first < y.first;         // 时间相同保持原始输入编号顺序
});
```

本页要点

用 stable_sort 且比较器只比 x.first < y.first，相等时自动保持原序。

例 1 | 排队接水：完整代码（2/2）

```
    long long waited = 0, clock_sum = 0;
    for (int i = 0; i < n; ++i) {
        if (i) cout << ' ';
        cout << a[i].second;
        waited += clock_sum;              // 第 i 个人需要等待前面所有人的总时间
        clock_sum += a[i].first;
    }
    cout << '\n' << fixed << setprecision(2) << (double)waited / n << '\n';
    return 0;
}
```

本页要点

输出排列时用 `if (i) cout << ' ';` 控制相邻空格，行末不留多余空格。

例 1 | 排队接水：易错对照与三组自测边界

- 常见错因 **1**（**WA**）：`waited` 开成 `int`。$n = 1000, T_i = 10^4$ 时总和达 5×10^9，溢出必错。
- 常见错因 **2**（**WA**）：先做了 `clock_sum += a[i].first` 再加进 `waited`，把本人接水时间算成等待。
- 常见错因 **3**（**WA**）：用普通 `sort` 且没比编号，相等元素顺序可能被打乱。

三组自测边界数据

1. 单人接水：`1 / 100` $\Rightarrow$ `1` 和 `0.00`。
2. 全等耗时：`3 / 5 5 5` $\Rightarrow$ `1 2 3` 和 `5.00`。
3. 最坏防溢出：`1000` 个 `10000` $\Rightarrow$ 平均等待 `4995000.00`。

例 2 | 活动选择（一本通 1323）[基础]：先读懂要求

大白话题意

学校有 n 个活动想使用同一间大礼堂，第 i 个活动占用时段 $[begin_i, end_i]$。同一时间只能办一个活动，但前一个结束时刻等于后一个开始时刻（如 $[1,4]$ 与 $[4,7]$）允许相邻安排。求最多能安排多少个活动？

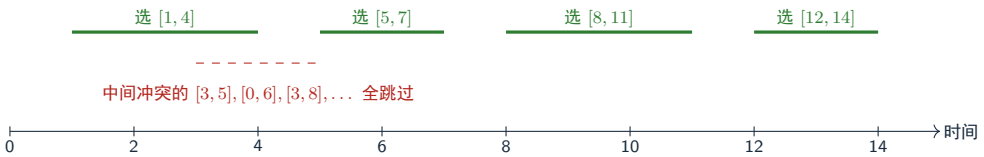
- 输入：第一行 $n \le 1000$；接下来 n 行每行两个整数 $begin_i < end_i \le 32767$。
- 处理：把所有区间按结束时间 end_i 升序排序，依次贪心挑选相容区间。
- 输出：一个整数，表示最多能安排的活动个数。
- 原题样例：11 个活动（`3 5，1 4，12 14，8 12，...`）$\Rightarrow$ 输出 `4`。

本页要点

看到区间选最多不重叠个数，立刻按“结束时间早到晚”排！

例 2 | 活动选择：怎么想到的与建模图解

- 按结束时间从小到大排好这 11 个样例区间：[1,4]，[3,5]，[0,6]，[5,7]，[3,8]，[5,9]，[6,10]，[8,11]，[8,12]，[2,13]，[12,14]。



本页要点

把每个活动画在数轴上：每选定一条绿色线段，就把它的右端点作为下一条线段的最低起点！

例 2 | 活动选择：原题样例手推表

按 *end* 排序后的 11 个活动，初始 `last_end = INT_MIN`，`chosen = 0`：

序号	区间 $[b, e]$	$b \geq$ `last_end`?	`chosen`	新 `last_end` / 说明
1	$[1, 4]$	真（$1 \geq -\infty$）	1	4（入选第 1 场）
$2 \sim 3$	$[3, 5], [0, 6]$	假（$3 < 4,\ 0 < 4$）	1	4（冲突跳过）
4	$[5, 7]$	真（$5 \geq 4$）	2	7（入选第 2 场）
$5 \sim 7$	$[3, 8], [5, 9], [6, 10]$	假（均 < 7）	2	7（冲突跳过）
8	$[8, 11]$	真（$8 \geq 7$）	3	11（入选第 3 场）
$9 \sim 10$	$[8, 12], [2, 13]$	假（$8 < 11,\ 2 < 11$）	3	11（冲突跳过）
11	$[12, 14]$	真（$12 \geq 11$）	**4**	14（扫完退出，输出 **4**）

本页要点

11 个区间按右端点排序后一趟扫过，干脆利落选出 4 个互不冲突的活动。

例 2 | 活动选择：核心步骤与相对模板的变化

按顺序计算	纸上得到	核对问题
1. 反转存入 pair	读入 b，e 存成 {e，b}	sort 会按 first 升序排吗？是的！
2. 初始化指针	chosen = 0，last_end = INT_MIN	允许活动在 0 时刻开始吗？允许！
3. 贪心扫描	若 b >= last_end 就选上	端点相连 $[1,4][4,7]$ 写 >= 而不是 >

相对模板改了哪里

这题就是第 3 节区间调度模板的直接原型：利用 pair<int,int> 先存结束时间再存开始时间，省去自定义 cmp，再用结构化绑定 for（auto [e，b] : act）简洁扫描。

本页要点

用 cin >> act[i].second >> act[i].first; 读入时直接反转，代码既短又不易出错。

例 2 | 活动选择：完整代码（1/2，code/activity.cpp）

```cpp
int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int n;
    if (!(cin >> n)) return 0;
    vector<pair<int, int>> act(n);       // {end, begin}
    for (int i = 0; i < n; ++i)
        cin >> act[i].second >> act[i].first;
    sort(act.begin(), act.end());        // 默认优先按结束时间升序
```

本页要点

输入格式依然是先读开始时间再读结束时间，只不过存到 second 和 first。

例 2 | 活动选择：完整代码（2/2）

```
    int chosen = 0, last_end = INT_MIN;
    for (auto [e, b] : act) {
        if (b >= last_end) {            // 端点相接 [1,4] 与 [4,7] 也合法
            ++chosen;
            last_end = e;
        }
    }
    cout << chosen << '\n';
    return 0;
}
```

本页要点

时间复杂度 $O(n \log n)$，空间复杂度 $O(n)$，在 $n \le 1000$ 下不到 1 毫秒完成。

例 2 | 活动选择：易错对照与三组自测边界

- 常见错因 **1**（**WA**）：用 `cin >> act[i].first >> act[i].second;` 直接按开始时间排序。
- 常见错因 **2**（**WA**）：判断相容写成 `b > last_end`，把 $[1,4],[4,7]$ 误杀掉。
- 常见错因 **3**（**WA**）：部分同学尝试“如果当前冲突但新活动结束更早则反悔替换”，其实提前排过序后不会出现后面结束更早的情况。

三组自测边界数据

1. 只有一个活动：`1 / 0 5` ⇒ 输出 1。
2. 端点连续相接：`3 / 0 2 / 2 4 / 4 6` ⇒ 输出 3。
3. 全包含在最大区间内：`3 / 1 100 / 2 3 / 4 5` ⇒ 跳过 `[1,100]` 选后两个，输出 2。

例 3 | 均分纸牌（一本通 1320）[基础]：先读懂要求

大白话题意

n 堆纸牌排成一排，第 i 堆有 a_i 张（保证总牌数是 n 的倍数）。
每次操作可以抓起某一堆的任意多张纸牌，放到它左邻或右邻的一堆上（1 号只能和 2 号互移，n 号只能和 $n-1$ 号互移）。问最少移动多少次能使每堆牌数相等？

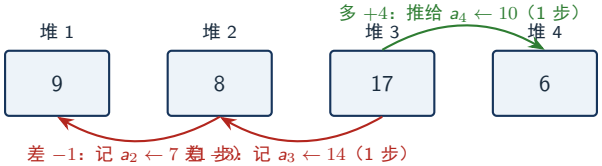
- 输入：第一行 $n \le 100$；第二行 n 个整数 $a_i \le 10000$。
- 处理：求出均值 avg，从左到右把每堆与 avg 的差直接划拨给右边相邻堆。
- 输出：一个整数，表示最少移动次数。
- 原题样例：`n=4`，四堆 `9 8 17 6` $\Rightarrow$ 输出 `3`。

本页要点

抓手在最左端第 1 堆：它只有一个邻居第 2 堆，差多少就非得在 1 和 2 之间搬一次不可！

例 3 | 均分纸牌：怎么想到的与建模图解

- 为什么从左往右“借负账”也完全等价？第 1 堆当前有 9 张（目标 $avg = 10$），它必须从第 2 堆借 1 张。哪怕第 2 堆此刻只有 8 张、被借完变成 7 张（甚至某题里变成负数），只要我们把“搬运顺序”在心里倒过来——先让第 3 堆给第 2 堆送牌、再让第 2 堆给第 1 堆送 1 张——总搬运次数完全一样！



本页要点

每堆（除了最后一堆）只要当前不等于 avg，就和右边做 1 次调整；一旦等于 avg 就跳过省 1 步！

例 3 | 均分纸牌：原题样例手推表

总和 $9+8+17+6=40$，平均数 $avg=40/4=10$。从 $i=0$ 到 $i=2$ 依次抹平：

位置 i	$a[i]\neq 10$?	差额 $d=a[i]-10$	操作后四堆数组	累计步数 `moves`
初始	—	—	`[9, 8, 17, 6]`	0
0	真（$9\neq 10$）	-1	`[10, 7, 17, 6]`	1（$a[1]+=-1$）
1	真（$7\neq 10$）	-3	`[10, 10, 14, 6]`	2（$a[2]+=-3$）
2	真（$14\neq 10$）	$+4$	`[10, 10, 10, 10]`	**3**（$a[3]+=4$）
3	终止（$i+1<4$ 假）	0	`[10, 10, 10, 10]`	退出，输出 **3**

本页要点

当把前 $n-1$ 堆都调成 avg 后，最后一堆因为总和守恒必然自动等于 avg！

例 3 | 均分纸牌：核心步骤与相对模板的变化

按顺序计算	纸上得到	核对问题
1. 求总和与均值	`sum` = $\sum a[i]$，`avg = sum / n`	总和是否可能较大？用 `long long sum`
2. 扫到 $n-2$ 下标	`for (i = 0; i + 1 < n; ++i)`	最后一堆 $n-1$ 需要再往右移吗？不用！
3. 判不等则转移	`if (a[i] != avg)` 更新 `a[i+1]`	正负差都只需 1 次操作，切勿重复计数

相对模板改了哪里

这题不用先排序（堆的位置固定不能乱换），而是“固定顺序的流式贪心”：用单向传递的视角证明第 i 堆只能靠第 $i+1$ 堆补齐，允许临时出现负差额。

本页要点

如果中间有某堆本来就凑巧等于 avg（如 `[10, 10, 10]`），`if (a[i] != avg)` 跳过它就省下了一步！

例 3 | 均分纸牌：完整代码（1/2，code/cards.cpp）

```cpp
int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int n;
    if (!(cin >> n)) return 0;
    vector<int> a(n);
    long long sum = 0;
    for (int &x : a) {
        cin >> x;
        sum += x;
    }
    int avg = (int)(sum / n), moves = 0;
```

本页要点

用范围 `for (int &x : a)` 边读入边累加总和，简洁不易写错下标。

例 3 | 均分纸牌：完整代码（2/2）

```
    for (int i = 0; i + 1 < n; ++i) {
        if (a[i] != avg) {
            a[i + 1] += a[i] - avg;      // 正负都只需一步搬运，允许暂存负差额
            ++moves;
        }
    }
    cout << moves << '\n';
    return 0;
}
```

本页要点

循环条件写成 `i + 1 < n`，避免 `a[i + 1]` 访问越界！

例 3 | 均分纸牌：易错对照与三组自测边界

- 常见错因 **1**（**WA**）：不敢允许 `a[i+1]` 变负，非要先找正数堆再折返，写得极其复杂还容易算多步数。
- 常见错因 **2**（**RE**）：循环写成 `i < n`，导致访问 `a[n]` 越界崩溃。
- 常见错因 **3**（**WA**）：中间有前缀和刚好等于 $k \times avg$ 时（如 `[12, 8, 12, 8]` 均值 10），把两边分成独立两组，代码里只要 `a[i] == avg` 不计步数就会自动断开（第 1 堆给第 2 堆 2 张后，$a[1]$ 变成 10 自动跳过！）。

三组自测边界数据

1. 本身已均分：`3 / 5 5 5` $\Rightarrow$ 输出 `0`。
2. 分成两段独立平衡：`4 / 12 8 12 8` $\Rightarrow$ 输出 `2`（第 2 堆与第 3 堆之间不搬运！）。
3. 只有 1 堆：`1 / 42` $\Rightarrow$ 输出 `0`。

例 4 | 删数问题（一本通 1321）[提高]：先读懂要求

大白话题意

给定一个不超过 240 位的正整数 n（不含前导零）和删除个数 s（$1 \le s < |n|$）。
从 n 中删去任意 s 个数字，剩下的数字按原来的左右顺序拼成新整数。要求新整数尽量小。
若删完后全为 0 输出 0，否则去掉多余前导零。

- 输入：第一行大整数字符串 n（最长 240 位）；第二行要删的位数 s（代码变量记作 k）。
- 处理：用单调递增栈从左往右扫，遇前大后小立刻删前面大数；不够删再从末尾补删。
- 输出：去除前导零后的最小非负整数。
- 原题样例：175438 删 4 位 $\Rightarrow$ 输出 13。

本页要点

240 位远超第 3 章讲过的 long long（仅 19 位）！必须直接用 string 读入！

例 4 | 删数问题：怎么想到的与建模图解

- 核心性质：比较两个位数相同的正整数大小，最高位数字小 **1**，低位再大也追不上！
- 所以只要从左往右看到“前一位数字 $>$ 后一位数字”（形成一个下坡拐点），把前面那个大数字删掉，后面的小数字立刻补到高位，结果严格变小！



本页要点

先删“下坡拐点”让结果单调不降；若扫完还没删够 k 个，剩下串已单调不降，从最右边尾巴删最大数字！

例 4 | 删数问题：原题样例手推表

对 "175438"（删 $k = 4$ 位），维护结果字符串 st 当作单调栈：

读入字符 c	栈顶 $>$ c 且 $k > 0$?	弹栈动作	操作后 st	剩余 k
'1'	假（栈空）	直接压入 '1'	"1"	4
'7'	假（$1 > 7$ 假）	直接压入 '7'	"17"	4
'5'	真（$7 > 5$）	弹 '7'（$k \to 3$），压入 '5'	"15"	3
'4'	真（$5 > 4$）	弹 '5'（$k \to 2$），压入 '4'	"14"	2
'3'	真（$4 > 3$）	弹 '4'（$k \to 1$），压入 '3'	"13"	1
'8'	假（$3 > 8$ 假）	压入 '8' 后得 "138"	"138"	1
末尾补删	真（$k = 1 > 0$）	弹出末尾 '8'（$k \to 0$），退出	**"13"**	0

本页要点

为什么不能只删一次？因为遇到 '3' 时可能会连续弹出前面好几个比它大的栈顶，所以要用 while！

例 4 | 删数问题：核心步骤与相对模板的变化

按顺序计算	纸上得到	核对问题
1. 字符串读大数	`string s, st; int k;`	千万别用 `long long`（240 位必溢出）
2. 单调递增栈消逆序	`while (k>0 && !st.empty() && st.back()>c)`	是不是写了 `while` 而不是 `if`？
3. 末尾补删与去零	补删至 $k = 0$；找首个非 `'0'` 下标	如果剩下全 `"000"`，不能输出空串要输出 `"0"`

相对模板改了哪里

这题是“栈式贪心”的典型代表：用普通 `string st` 直接充当单调栈（支持 `back()`, `push_back()`, `pop_back()`），最后利用 `find_first_not_of('0')` 一步去掉前导零。

本页要点

用 `string` 本身当栈的好处是遍历完就能直接切片输出，完全不用再把栈倒出来！

例 4 | 删数问题：完整代码（1/2，code/delete_digits.cpp）

```cpp
int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    string s; int k;
    if (!(cin >> s >> k)) return 0;
    string st;
    for (char c : s) {
        while (k > 0 && !st.empty() && st.back() > c) {
            st.pop_back();              // 遇到前大后小，立刻删去高位大数
            --k;
        }
        st.push_back(c);
    }
```

本页要点

每个字符至多入栈、出栈各一次，总时间依然是线性的 $O(|s|)$（头文件同前）。

例 4 | 删数问题：完整代码（2/2）

```
    while (k > 0 && !st.empty()) {
        st.pop_back();
        --k;                                    // 本身递增时从低位末尾补充删
    }
    size_t p = st.find_first_not_of('0'); // 去掉前导 0，全 0 则输出 0
    cout << (p == string::npos ? "0" : st.substr(p)) << '\n';
    return 0;
}
```

本页要点

`st.find_first_not_of('0')` 找不到非零字符时返回 `string::npos`，此时必须打印 `"0"`。

例 4 | 删数问题：易错对照与三组自测边界

- 常见错因 **1**（**WA**）：用 `long long` 读 n，240 位整数直接溢出。
- 常见错因 **2**（**WA**）：漏掉末尾补删 `while (k > 0)`，导致输入 12345 删 2 位时一位没删。
- 常见错因 **3**（**WA**）：忘记去前导零，或者去零后空串没有特判 `"0"`（如 10023 删 1 位误出 0023，1000 删 1 位误出空行）。

三组自测边界数据

1. 删后带前导零：`10023 / 1` $\Rightarrow$ 删 `'1'` 剩 `"0023"` $\rightarrow$ 输出 23。
2. 删完剩全零：`1000 / 1` $\Rightarrow$ 删 `'1'` 剩 `"000"` $\rightarrow$ 输出 0。
3. 原串严格递增：`12345 / 2` $\Rightarrow$ 从末尾删去 `'5','4'` $\rightarrow$ 输出 123。

贪心策略与邻项交换高频错误对照表

题号 / 场景	常见错误写法	正确做法与原因
活动选择（1323）	按开始时间早或时长短排序	只有按结束时间早排序才能留给后面最大空地
排队接水（1319）	把本人接水耗时也算进等待时间	`waited` 只累加自己前面所有人的用时总和
同耗时排序稳定性	同耗时用普通 `sort` 且不比编号	要求保留原编号小在前，改用 `stable_sort`
均分纸牌（1320）	不敢记负账强行左右搜索	按顺序把差额直接划给下一堆，允许暂存负差额
删数问题（1321）	`long long` 读数或忘去前导零	用 `string` 读 240 位，补删末尾并判全零输出 0

本页要点

能写出邻项交换不等式或替换证明才动手敲代码；拍脑袋写贪心最容易全盘 WA。

评测结果定位指南（CE / WA / RE / TLE）

评测结果	本章最常见元凶	第一时间排查步骤
CE	漏 `<queue>` / `<iomanip>` / `<algorithm>`	检查 `priority_queue`、`setprecision`、`sort` 头文件
WA	贪心规则本身不成立；漏 `>=`；溢出 `int`；未去前导零	立刻构造 $3 \sim 4$ 组逆序/交叉/全等小数据找反例
RE	比较器写了 `<=` 违背严格弱序；小根堆只剩 1 堆连 `pop` 两次	比较器改严格 `<`；合并循环写 `pq.size() > 1`
TLE	暴力枚举 2^n 或 $n!$；没用堆而每轮重新完整 `sort`	用排序一次性有序或小根堆 $O(n \log n)$ 维护

本页要点

贪心绝大部分报错都是 WA——一旦 WA 别急着改语法，先拿笔在草稿纸上构造反例！

提交前检查清单（六必查）

1. 是否用“邻项交换不等式”或“替换剪贴法”验证过当前策略，并试着造过 2 组反例？
2. 自定义 `cmp` 函数是否严格写了 `<`（或 `>`），绝对没有写 `<=` 或 `>=`？
3. 当排序关键字相等时（如接水耗时相同），有没有用 `stable_sort` 或第二关键字保序？
4. 区间问题中端点恰好相连（$[1,4]$ 与 $[4,7]$）是相容还是冲突？写的是 `>=` 还是 `>`？
5. 所有累计总等待时间、总合并体力、乘积比较是否都已开成 `long long`？
6. 数字串问题（删数）是否用 `string` 读入，并在去前导零后处理了全零输出 `"0"`？

最后一步

提交前亲自造三组小数据：全相等、严格逆序、跨头跨尾交叉区间，手算核对后再交！

本章全部练习与完成标准

1. 排队与邻项交换：
   - 排队接水（一本通 1319，`code/water_queue.cpp`）、接水问题（一本通 1233）
   - 国王游戏（NOIP 2012，结合第 3 章高精度乘除法）
2. 区间贪心：
   - 活动选择（一本通 1323，`code/activity.cpp`）、整数区间（一本通 1324）
3. 哈夫曼合并、纸牌与栈式贪心：
   - 合并果子（经典小根堆）、均分纸牌（一本通 1320，`code/cards.cpp`）、删数问题（一本通 1321，`code/delete_digits.cpp`）

完成标准

能默写《排队接水》和《国王游戏》的邻项交换不等式；能说出活动选择为什么按右端点排；独立 AC 上述精选习题。

课后安排与下一章预告

- 复习：合上课件，独立证明《排队接水》为什么按 T_i 升序、《活动选择》为什么不能按时长短排。
- 练习：把 code/ 下 4 份源码自己敲一遍，重点测《删数问题》的 10023、1000 和严格递增用例，再额外 AC《整数区间》（1324）与《接水问题》（1233）。
- 下一章预告：第 6 章《递推与递归》——当遇到像硬币找零、0-1 背包那样“局部贪心会翻车”的问题时，怎么从小状态一步步推到大状态（递推），又怎么把大问题拆成相同小问题并加上“记忆化”避免重复计算！

本章一句话

写贪心前先动笔比一比“相邻交换”，不等式成立了再放心排序。

谢谢！