基础算法 · 第六章

贪心算法

每一步选眼前最好，凭什么全局最优？

HGNU-ACM 实验室算法组

截图配套：6 道例题，11 道练习题





本章目录 | 前半

贪心从哪里来

排序之后逐个选

均分与传送

删数问题

本章目录｜后半

调度与覆盖

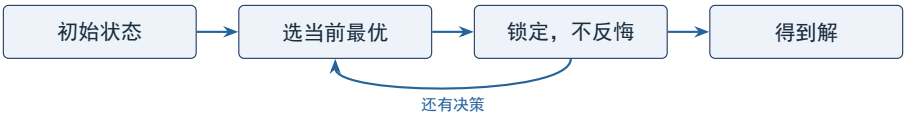
区间上的贪心

易错点与模板

本章全部练习题索引

贪心：每一步都选“眼前最好”

- 贪心算法把求解过程拆成一串决策：每一步只利用已经知道的信息，在候选方案中选当前看起来最好的一个。
- 选定之后不反悔、不回退，继续处理剩下的部分；代码通常很短，排序后线性扫描，或者用堆、栈维护当前最优。
- 真正的难点不是写代码，而是回答：眼前最好，为什么能推出全局最好？



本页要点

贪心是一种“有证明的局部最优策略”，不是凭感觉随便选。

一个完整过程：面值 $1, 5, 10, 20, 50, 100$ 找零 63 元

步骤	还需支付	选择面值	剩余
1	63	50	13
2	13	10	3
3	3	1	2
4	2	1	1
5	1	1	0

- 每一步都选“不超过剩余金额的最大面值”，5 步结束。
- 这个面值系统下贪心最优；换一套面值就未必（后面给出反例）。
- 描述贪心过程的三要素：当前状态（剩余金额）、候选集合（面值）、选择关键字（面值最大）。

本页要点

把“当前状态—候选集合—选择关键字”写清楚，贪心过程就描述完了。

一个贪心选择要同时过两关

可行 + 可证最优 正确贪心	可行 + 未证最优 可能 WA
不可行 方案本身违反约束	最优但不可行 没有意义

- 可行性：选择必须满足题目约束（容量、时间、相邻关系）。
- 最优性：还要证明“选它不会错过最优解”，否则只是可行解。
- 只过第一关，可能得到一个能运行但非最优的方案；两关都过，才是贪心算法。

本页要点

先检查可行，再证明最优；顺序反了容易把错误策略写成代码。

什么时候能用（一）：贪心选择性质

- 定义：存在一个全局最优解，它包含当前这一步的贪心选择 g。
- 注意“存在”二字：不要求所有最优解都含 g，只要有一个含 g 即可。
- 验证方法：先在小数据上枚举最优解，看是否总能找到一个含 g 的解；再尝试用交换论证证明。



本页要点

贪心选择性质回答的是：第一步选 g，会不会把最优解排除掉；反例见后面的 $0/1$ 背包。

什么时候能用（二）：最优子结构

- 定义：选定 g 后，原问题剩下的部分是一个同类型的子问题；并且“g + 子问题的最优解”就是原问题的最优解。
- 直觉：第一步已经固定，后面只要把剩下的部分做到最好，整体就最好。
- 检查方法：把 g 删掉，看剩下的数据是否还在描述同一个问题。



两个性质的分工

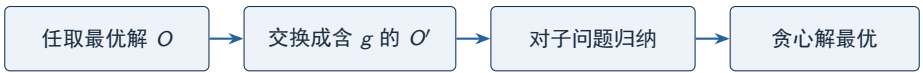
贪心选择性质保证“第一步不亏”；最优子结构保证“后面每一步都能继续贪”。

本页要点

没有最优子结构，贪心只能做一步，不能递归地做下去。

两个性质怎样配合：正确性证明模板

1. 贪心选择性质：任取一个最优解 O，用交换论证把它改造成包含 g 的最优解 O'。
2. 最优子结构：在 O' 中删掉 g，剩下部分是子问题的最优解；否则把它换成更好的子问题解，会得到比 O' 更好的原问题解，矛盾。
3. 归纳：对子问题重复第 1、2 步，最终得到“贪心解 $=$ 某个最优解”。

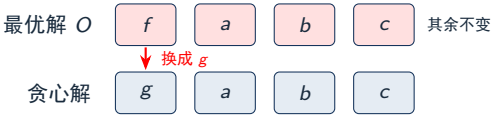


本页要点

交换论证负责“第一步”，最优子结构负责“每一步都能继续”，归纳负责收尾。

交换论证：把任意最优解改造成贪心解

1. 取任意一个最优解 O，从左到右找到它与贪心解第一个不同的位置。
2. 设 O 在这一步选了 f，贪心选的是 g。把 O 中的 f 换成 g，其余不变。
3. 证明换完之后仍然可行，并且目标值不变差（不增或不减，取决于问题）。
4. 重复以上步骤，直到 O 变成贪心解；于是贪心解也是最优解。



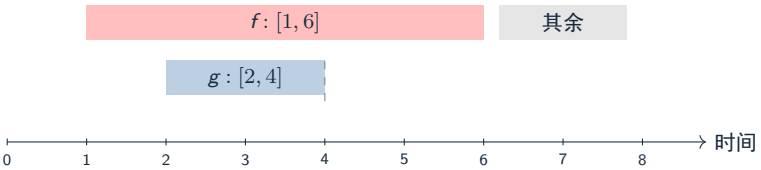
本页要点

交换论证的关键不是“换”，而是证明“换完仍可行、且不变差”。

交换论证实战：活动选择

设 g 是所有活动中结束最早的活动，某个最优解的第一个活动是 f。

- 因为 g 结束不晚于 f，把 f 换成 g 后，g 的结束时间更早，原来排在 f 后面的活动仍然与 g 相容。
- 活动个数不变，所以换完仍是最优解，而且它包含 g。
- 这就证明了贪心选择性质；剩下的活动仍然是“活动选择”子问题，满足最优子结构，可以继续贪心。



本页要点

“结束最早”之所以对，是因为它给后面留下的时间最多。

交换论证实战：排队接水

设相邻两人 x, y，x 在 y 前且 $T_x > T_y$。只交换这两人，其他人不变。

- 交换前，这两人的接水时间对总等待的贡献为 $(n-k)T_x + (n-k-1)T_y$。
- 交换后，贡献变为 $(n-k)T_y + (n-k-1)T_x$。
- 差值 $= T_y - T_x < 0$，所以交换后总等待更小，原顺序不可能最优。
- 因此最优顺序中不存在“前一个接水时间更长”的相邻逆序，按 T_i 升序排序即可。

这就是“相邻交换”套路

很多排序贪心都能这样证：假设存在逆序，交换后目标函数变好，矛盾。

本页要点

排序关键字不是猜出来的：让“交换后变好”的那个量，就是关键字。

反例意识：局部最优不等于全局最优

反例 1：硬币找零

面值 $1, 5, 11$，凑 15。面值大优先：$11+1+1+1+1$，共 5 枚；最优 $5+5+5$，共 3 枚。

反例 2：按开始时间选活动

活动 $[0,10], [1,2], [2,3]$。选最早开始的 $[0,10]$ 后只能再选 0 个；不选它可安排 $[1,2], [2,3]$，共 2 个。

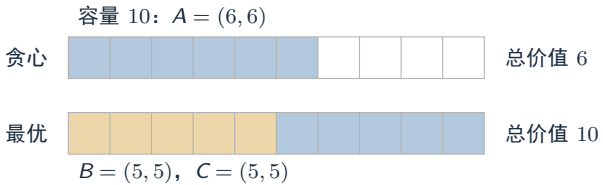
反例 3：$0/1$ 背包

容量 10，物品 $(6,6), (5,5), (5,5)$。先拿价值 6 的物品，剩余容量 4，总价值 6；最优 $5+5=10$。若物品可分割，贪心又正确。

本页要点

检验贪心最快的方法：专门构造反例攻击你准备使用的排序关键字。

反例图解：0/1 背包与分数背包



- 0/1 背包：物品不可分割，按“价值最大”先拿 A，剩下容量 4 浪费，总价值 6；选 $B + C$ 总价值 10。
- 分数背包：物品可以切开，先装单位价值最高的，装满为止；这时贪心选择性质成立。
- 同一个故事，约束变了，贪心的正确性就变了。

本页要点

先确认题目允许“分割/重复使用”还是“只能选一次”，再决定能不能贪。

怎样主动找反例：攻击排序关键字

1. 列出所有可能的关键字：价值、重量、单位价值、开始时间、结束时间、持续时间。
2. 用两个元素做相邻交换，看目标函数会不会变好；会变好就说明关键字不对。
3. 构造“一个特别大的 $+$ 许多小的”“全部相等”“两个关键字相同”的数据。
4. 小数据枚举所有方案，与贪心结果对拍；找到第一组不同就是反例。

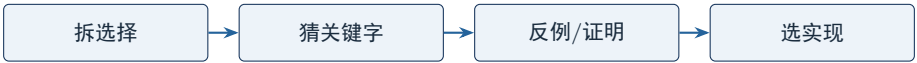
候选关键字	看似合理	反例长相
价值降序	先拿最值钱的	大价值占满容量，两个中价值更优
重量升序	先拿最轻的	轻物品价值低，浪费容量
单位价值降序	性价比最高	$0/1$ 背包中装不满，剩余容量浪费

本页要点

反例不需要复杂：两三个物品、容量刚好差一点，通常就能击穿错误关键字。

贪心解题四步

1. 拆选择：每一步到底在选什么？一个人、一个活动、一个数字，还是一条边界？
2. 猜关键字：当前最好的标准是什么？时间短、结束早、单位价值高、右端点小？
3. 找反例/证明：先构造小数据攻击关键字；找不到反例，再用交换论证或归纳证明。
4. 选实现：排序后扫描、优先队列、单调栈，或直接模拟。



以活动选择为例

选择对象是“下一个活动”；关键字是“结束时间最早”；证明用交换论证；实现是排序后扫描，维护 `last_end`。

本页要点

四步的顺序不能反：先有选择和关键字，再谈证明与实现。

贪心与动态规划：别硬贪

	贪心	动态规划
每步保留	只保留一个选择	保留所有可能状态
正确性	必须证明局部最优可推出全局最优	转移覆盖所有情况即可
典型复杂度	常为 $O(n \log n)$	常为 $O(n^2)$ 或更高
典型例子	分数背包、活动选择、删数	$0/1$ 背包、最大子矩阵

- 贪心每一步只保留一个选择，所以必须证明“这个选择不亏”。
- 动态规划保留所有可能状态，转移覆盖所有情况即可，不需要交换论证。
- 本章练习里的“最大子矩阵”名字看不出算法，它实际用的是压缩列之后的一维最大子段和。
- 如果找不到贪心证明，先考虑动态规划或搜索，不要硬套。

本页要点

分界线：局部选择能不能被证明“总是不亏”。

常见套路总览：四类贪心

排序后扫描 先排序，再线性扫描	区间调度 按端点排序，维护边界
优先队列 在线决策，堆顶最优	单调栈 维护单调性，弹出无用元素

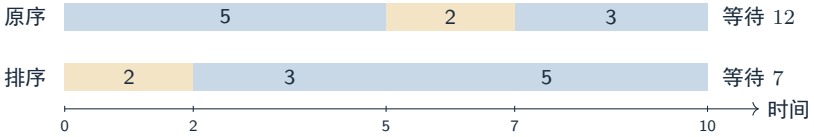
- 看到“先全部读入再安排”，优先想排序；看到“按到达顺序立即决定”，优先想堆。
- 看到“区间不重叠/选点覆盖”，优先想按端点排序；看到“保留字典序最小”，优先想单调栈。

本页要点

套路决定实现，证明决定正确性；两者不能互相替代。

套路一：排序后贪心

- 适用条件：所有数据可以先读进来，选择之间没有“到达顺序”约束；目标函数能写成“每个元素乘以一个位置权重”的形式。
- 做法：按关键字排序，然后从左到右扫描，维护一个状态（如 `last`）。
- 证明：相邻交换。若存在逆序，交换后目标函数变好，矛盾。



典型题

排队接水（时间升序）、书架（高度降序）、金银岛（单位价值降序）。

本页要点

排序关键字要让“交换后变好”成立；否则排序再漂亮也没用。

套路二：区间调度

- 适用条件：每个对象占用一段连续区间，约束是“不能重叠”或“必须被覆盖”。
- 做法：按左端点或右端点排序，扫描时维护“上一个选择的结束位置”或“最后选中的点”。
- 关键：先明确优化目标——最多选几个、最少用几个点，还是最多重叠几层。
- 三类目标：最多不重叠、最少点覆盖、最大重叠层数。

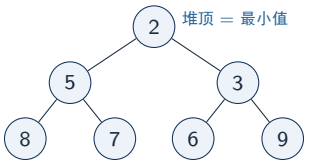


本页要点

区间贪心的核心不是“区间”，而是“你要优化的量”。

套路三：优先队列维护当前最优

- 适用条件：数据按顺序到达，必须立即决策；候选集合会动态变化。
- 做法：把候选放进小根堆，堆顶就是当前最小（或最大）的候选；取出、更新、再放回。
- 复杂度：每次操作 $O(\log n)$，总共 $O(n \log n)$。
- 典型题：接水问题、合并果子、拦截导弹在线贪心。

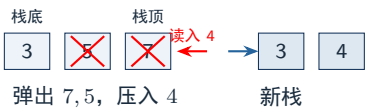


本页要点

堆里存的不是“所有数据”，而是“当前还有用的候选”。

套路四：单调栈保留“有用”的元素

- 适用条件：从左到右处理序列，当前元素会“淘汰”前面某些元素；被淘汰的元素以后不可能再成为答案。
- 做法：用栈保存候选，保持栈内单调；新元素到来时，不断弹出违反单调性的栈顶，再压入新元素。
- 复杂度：每个元素最多进栈一次、出栈一次，总共 $O(n)$。



典型题

删数问题（删除下降点）、下一个更大元素、柱状图最大矩形。

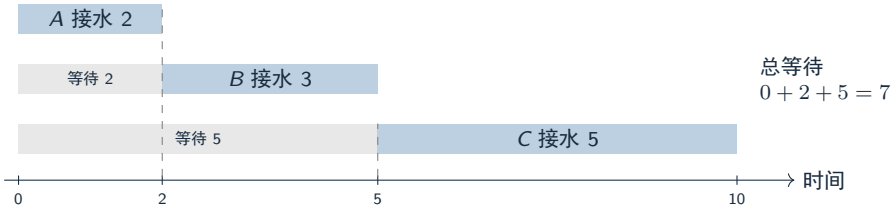
本页要点

单调栈的本质：用 $O(n)$ 的时间，删掉永远不会再被用到的候选。

排序贪心：让总等待时间最小

设有 n 个人接水，第 i 个人需要 T_i 秒。若接水顺序为 $p_1, p_2, \ldots, p_n$，则总等待时间 $W = \sum_{k=1}^{n} (n-k)\, T_{p_k}$，排得越前的位置权重越大。

- 一个人的接水时间会贡献给所有排在他后面的人，所以短作业应放前面。
- 由排序不等式，应把接水时间短的人放在权重最大（靠前）的位置。



本页要点

排序关键字：T_i 升序；时间相同按原始编号稳定输出。

例 6.1 | 排队接水：题意与样例

题意

n 个人在一个水龙头前排队，第 i 个人接水需要 T_i。求一种排队顺序，使所有人的平均等待时间最小。输出顺序与平均等待时间（两位小数）。

样例：$T = (56, 12, 1, 99, 1000, 234, 33, 55, 99, 812)$。按时间升序排列后：

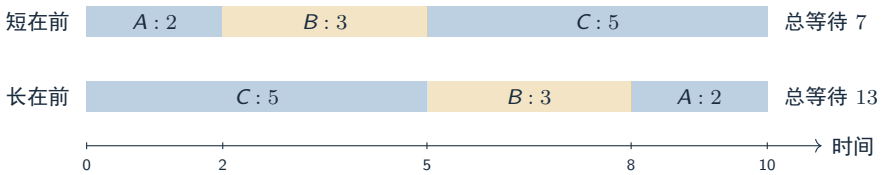
编号	3	2	7	8	1	4	9	6	10	5
T	1	12	33	55	56	99	99	234	812	1000
等待	0	1	13	46	101	157	256	355	589	1401

总等待 $= 2919$，平均 $= 2919/10 = 291.90$。两个 99 按输入编号 $4, 9$ 排列。

本页要点

平均等待不是总完成时间：第一个人的等待时间固定为 0。

例 6.1 | 为什么“短作业优先”是对的



- 顺序 $2,3,5$：总等待 $0+2+5=7$。
- 顺序 $5,3,2$：总等待 $0+5+8=13$。
- 把长作业提前，会让后面所有人都多等一段；短作业提前则相反。

本页要点

相邻交换的结论：只要存在 $T_x > T_y$ 且 x 在 y 前，就还能改进。

例 6.1 | 完整程序（上）：读入并排序

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

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int n;
    cin >> n;
    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 pair<int, int>& x, const pair<int, int>& y) {
            return x.first < y.first; // 时间升序，相同保持原序
        });
```

例 6.1 | 完整程序（下）：累计等待并输出

```
    long long waited = 0; // 总等待时间
    long long clock = 0; // 当前时间
    for (int i = 0; i < n; ++i) {
        if (i > 0) {
            cout << ' ';
        }
        cout << a[i].second;
        waited += clock;
        clock += a[i].first;
    }
    cout << '\n';
    cout << fixed << setprecision(2) << (double)waited / n << '\n';
    return 0;
}
```

- 上一页与本页合起来是一个完整程序。
- 先输出编号，再把当前时间累加进总等待，最后让当前时间前进。

例 6.1 | 复杂度与易错点

- 时间复杂度 $O(n \log n)$，空间复杂度 $O(n)$。
- 平均等待时间用 `long long` 保存总等待，最后再除以 n；$n = 1000$ 时总和仍可能较大，不要用 `int` 中间量。
- 第一个人的等待时间是 0，从第二个人开始累加前面的接水时间。
- 时间相同的人顺序不影响平均等待，但会影响输出顺序；用 `stable_sort` 或把编号作为第二关键字。
- 输出必须是两位小数：`fixed << setprecision(2)`。

变式：多个水龙头

若同时有 m 个水龙头，就不能简单排序了。按原顺序依次把人放到“最早空出来的水龙头”上，用小根堆维护即可，这就是练习中的“接水问题”。

本页要点

排队接水是“单机排序”，接水问题是“多机调度”，两者都用到了短作业优先。

均分纸牌：从左到右“欠债还钱”

n 堆纸牌排成一列，总张数是 n 的倍数。设

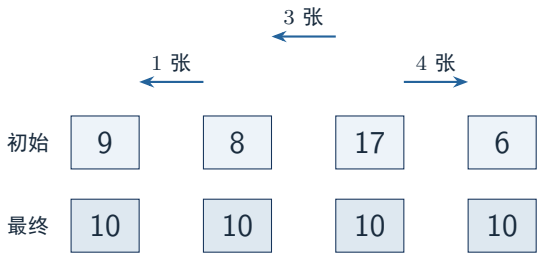
$$\bar{a} = \frac{1}{n}\sum_{i=1}^{n} a_i, \qquad d_i = a_i - \bar{a}.$$

- 前 i 堆一共“多出”$S_i = d_1 + d_2 + \cdots + d_i$ 张。
- 这些多出（或缺少）的牌，只能通过第 i 堆与第 $i+1$ 堆之间的边界运输。
- 因此只要 $S_i \neq 0$，这条边界就至少要被使用一次；下界是 $\#\{i < n : S_i \neq 0\}$。
- 从左到右扫描，每次把 $a_i - \bar{a}$ 直接交给右边，恰好让每条非零边界只用一次，达到下界。

本页要点

最少移动次数 $= \#\{i < n : S_i \neq 0\}$；一次可以搬多张，不是一张算一次。

均分纸牌 | 一列纸牌的搬运图



- 第 1 堆少 1 张，从第 2 堆拿 1 张；第 2 堆变成 7，再少 3 张。
- 第 3 堆多 7 张：先补第 2 堆 3 张，再给第 4 堆 4 张。
- 共 3 次。移动顺序可以不同，但最少次数固定。

本页要点

每次只处理当前最左边的不平衡堆，把它与右边一次结清。

例 6.2 | 样例手推与前缀和

样例

$n = 4$，纸牌数为 $9, 8, 17, 6$，平均数 $\bar{a} = 10$。

堆号 i	1	2	3	4
a_i	9	8	17	6
$d_i = a_i - 10$	-1	-2	7	-4
前缀和 S_i	-1	-3	4	0

1. 第 1 堆差 -1：从第 2 堆取 1 张，$[10, 7, 17, 6]$，移动 1 次。
2. 第 2 堆差 -3：从第 3 堆取 3 张，$[10, 10, 14, 6]$，移动 2 次。
3. 第 3 堆差 4：把 4 张给第 4 堆，$[10, 10, 10, 10]$，移动 3 次。

前缀和不为 0 的位置是 $1, 2, 3$，正好 3 个，与答案一致。

本页要点

实现时不必真的逐张模拟；只需判断 a_i 是否等于平均数。

例 6.2 | 完整程序（上）：读入并求平均

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

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int n;
    cin >> n;
    vector<int> a(n);
    long long sum = 0;
    for (int& x : a) {
        cin >> x;
        sum += x; // 总牌数
    }
    int avg = (int)(sum / n); // 题目保证整除
```

例 6.2 | 完整程序（下）：从左到右结清

```
    int moves = 0;
    for (int i = 0; i + 1 < n; ++i) {
        if (a[i] != avg) {
            a[i + 1] += a[i] - avg; // 把差额交给右边
            ++moves;
        }
    }
    cout << moves << '\n';
    return 0;
}
```

- 上一页与本页合起来是一个完整程序。
- 循环只处理前 $n-1$ 堆；最后一堆会被前面的操作自动补齐。

例 6.2 | 复杂度与易错点

- 时间 $O(n)$，空间 $O(n)$；也可以用前缀和直接统计，空间 $O(1)$。
- $n = 1$ 时不需要移动，循环不执行，输出 0。
- 题目保证总数是 n 的倍数，所以 `sum / n` 是精确的；若题目不保证，要先判断是否无解。
- 一次可以移动“若干张”，所以搬运 100 张也只算一次；不要写“逐张移动”的模拟。
- 差值可能为负，表示当前堆要从右边拿牌；`a[i+1] += a[i] - avg` 同时处理了两种方向。
- 总牌数可能较大，求和用 `long long`。

本页要点

关键不是模拟搬运，而是数清楚“哪些相邻边界必须被用到”。

删数问题：让高位尽可能小

给定一个高精度正整数 n（最多 240 位）和整数 s，删去 s 个数字，保持剩余数字的相对顺序，使得到的新数最小。

- 高位权重大于所有低位的总和：首位小 1，后面再大也补不回来。
- 从左到右找第一个下降点 $d_i > d_{i+1}$，删掉 d_i；删后可能产生新的下降点，继续找。
- 若整个串不下降，删除一定发生在末尾。

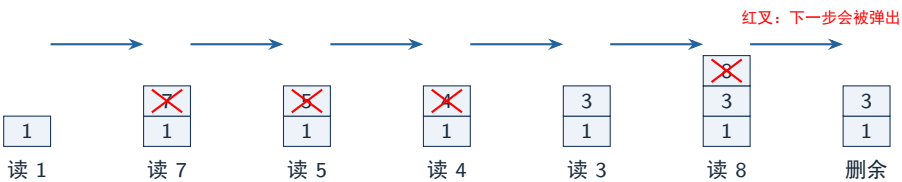


本页要点

删除第一个下降点左边的数字；若整个串不下降，就删末尾。

删数问题 | 单调栈怎样一遍扫完

- 栈内保存“目前已经确定”的前缀，始终保持不下降。
- 读入 c：只要栈顶比 c 大且还有删除次数，就弹栈；然后压入 c。
- 每个数字最多进栈、出栈一次，总时间 $O(|n|)$。



本页要点

“删下降点”一次只删一个；栈把“回头找下降点”变成了一遍扫描。

例 6.3 | 逐步手推

样例：$n = 175438$，$s = 4$。

步骤	读入	操作	栈（从底到顶）	剩余 s
1	1	压栈	1	4
2	7	压栈	1,7	4
3	5	弹掉 7	1,5	3
4	4	弹掉 5	1,4	2
5	3	弹掉 4	1,3	1
6	8	压栈	1,3,8	1
7	–	删末尾 8	1,3	0

结果为 13，与样例一致。

- 第 3 步删 7 后，1 成为最高位，5 成为第二位，比原来的 $17\ldots$ 更小。
- 最后串已经不下降，删除次数还没用完，只能从末尾删；因为末尾的权最小。

本页要点

每一步都在保证“当前前缀最小”，这就是删数问题的贪心选择性质。

例 6.3 | 完整程序（上）：单调栈删除

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

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    string s;
    int k;
    cin >> s >> k; // s 为高精度整数，k 为删除个数
    string st; // 单调不下降栈
    for (char c : s) {
        while (k > 0 && !st.empty() && st.back() > c) {
            st.pop_back(); // 栈顶更大，删掉它
            --k;
        }
        st.push_back(c);
    }
```

例 6.3 | 完整程序（下）：删剩余次数与前导零

```
    while (k > 0 && !st.empty()) { // 没有下降点，从末尾删
        st.pop_back();
        --k;
    }
    size_t p = st.find_first_not_of('0'); // 跳过前导零
    if (p == string::npos) {
        cout << 0 << '\n'; // 全部删完或全为零
    } else {
        cout << st.substr(p) << '\n';
    }
    return 0;
}
```

- 上一页与本页合起来是一个完整程序。
- 第二个 `while` 处理“整个串不下降”的情况，从末尾删。
- `find_first_not_of('0')` 去掉前导零；全为零时输出 0。

例 6.3 | 复杂度与易错点

- 时间 $O(|n|)$，空间 $O(|n|)$；每个数字最多进栈、出栈各一次。
- 不能简单地“每次删最大的数字”。例如 123546 删 1 位：删 6 得 12354，删 5 得 12346，后者更小；位置比数值大小更重要。
- 结果可能有前导零。例如 10200 删 1 位，删掉 1 得 0200，应输出 200，不是 0200。
- 若删除后为空（例如所有位都被删完），按“正整数”约定输出 0。
- 输入长度可达 240 位，必须用字符串，不能用 `long long`。
- 若整个串单调不降，删除会全部发生在末尾；所以循环结束后还要继续弹栈。

本页要点

先想清楚“比较两个候选数”的规则，再决定删除位置。

拦截导弹：一套系统就是一个不上升子序列

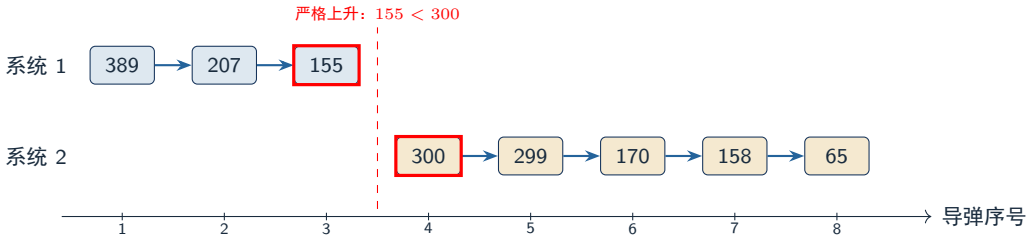
导弹依次飞来，一套系统的第一发可以打到任意高度，之后每一发都不能高于前一发。

- 一套系统能拦截的，是原序列的一个不上升子序列；问题变成用最少的不上升子序列覆盖整个序列。
- 在线贪心：维护每套系统“最后一发”的高度。新来 h，在所有能接住它的系统（最后一发高度 $\geq h$）中，选最后一发高度最小的那套。
- 若没有系统能接住 h，就新开一套。
- 结论（Dilworth 定理）：最少系统数等于最长严格上升子序列的长度。

本页要点

一套系统 $=$ 一个不上升子序列；答案 $=$ 最长严格上升子序列长度。

例 6.4 | 两套系统怎样分工



- 系统 1 接 $389, 207, 155$；第 4 枚 300 高于 155，接不上。
- 新开系统 2，接 $300, 299, 170, 158, 65$。
- 红框标出一对严格上升的导弹：它们不能由同一套系统拦截。

本页要点

不是按顺序轮流分配，而是每次选“最合适”的那套系统。

例 6.4 | 为什么答案等于最长严格上升子序列

- 若存在严格上升子序列 $h_{i_1} < h_{i_2} < \cdots < h_{i_k}$，其中任意两枚都不能由同一套系统拦截（同一套系统要求不上升）。所以系统数至少是 k。
- 反过来，按“最合适系统”贪心得到的系统数，恰好等于最长严格上升子序列长度。这是 Dilworth 定理在“不上升链覆盖”上的特例。
- 对大一新生，可以先记住结论，再用小数据验证；正式证明会在后续图论/偏序课程中遇到。

实现选择

$n \le 1000$，用 $O(n^2)$ 的 LIS 就足够：$\mathrm{dp}[i]$ 表示以第 i 枚结尾的最长严格上升子序列长度。也可以用贪心 + 二分做到 $O(n \log n)$。

本页要点

相等高度可以放进同一套系统，所以 LIS 必须严格上升，判断写作 $a_j < a_i$。

例 6.4 | 完整程序（上）：读到文件结束

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

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    vector<int> a;
    int x;
    while (cin >> x) { // 读到文件结束
        a.push_back(x);
    }
    int n = (int)a.size();
    vector<int> dp(n, 1); // dp[i]: 以 i 结尾的 LIS
    int ans = 0;
```

例 6.4 | 完整程序（下）：$O(n^2)$ 求 LIS

```
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < i; ++j) {
            if (a[j] < a[i]) { // 严格上升
                dp[i] = max(dp[i], dp[j] + 1);
            }
        }
        ans = max(ans, dp[i]);
    }
    cout << ans << '\n';
    return 0;
}
```

- 上一页与本页合起来是一个完整程序。
- 外层枚举结尾 i，内层枚举前驱 j；相等高度不能接，所以用 $<$。

例 6.4 | 复杂度与易错点

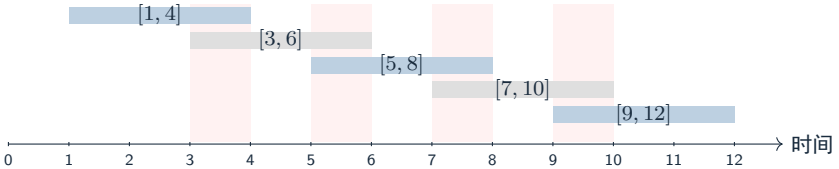
- $O(n^2)$ 时间、$O(n)$ 空间；$n \le 1000$ 时约 10^6 次比较，很快。若 n 到 10^5，改用贪心 + `lower_bound` 的 $O(n \log n)$ 写法。
- 输入没有先给 n，样例就是一行高度，读到文件结束；不要多读一个“导弹数量”。
- 相等高度属于“不高于”，可以放在同一套系统，所以 LIS 判断必须是 $a_j < a_i$，不能写 $a_j \le a_i$。
- 若用在线贪心实现，必须选择“能接住的最后一发高度中最小的”，随便选一套可能浪费高能力系统。
- 高度不超过 30000，用 `int` 即可。

本页要点

先判断题目要的是严格上升还是不下降，再把比较符号写对。

区间怎样表示：端点、相容与重叠

- 闭区间 $[l, r]$ 包含端点；半开区间 $[l, r)$ 不含右端点；开区间 (l, r) 两端都不含。
- 两个区间相容：一个的右端点不超过另一个的左端点（闭区间允许相等）。
- 区间问题先画时间轴：每条线段是一个区间，重叠部分就是冲突区域。



本页要点

做题第一步：把“开始/结束”画成线段，端点是否取等要标清楚。

区间问题的三种常见目标

目标	排序关键字	每步选择
最多不重叠区间	右端点升序	能接上就选
最少点覆盖所有区间	右端点升序	未覆盖就选右端点
最大重叠层数	左端点扫描	进入 $+1$，离开 -1

- 三个问题看起来都像“区间”，但目标不同，排序关键字和选择规则也不同。
- 扫描线遇到相同坐标时，先处理“离开”再处理“进入”，否则端点相接会被误算成重叠。
- 先写出目标函数，再决定按左端点还是右端点排序。

本页要点

不要背“区间题按右端点排”，要先看题目到底要优化什么。

活动选择：结束越早，留给后面的越多

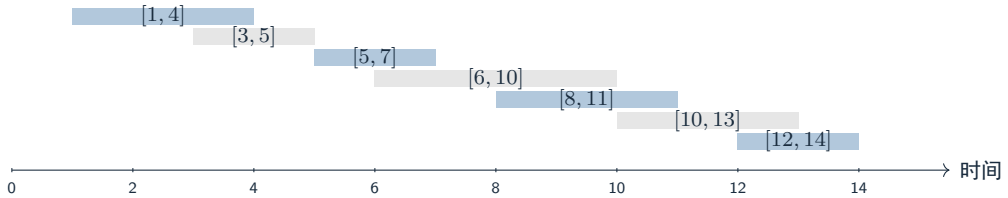
有 n 个活动都要用同一个礼堂，第 i 个活动占用 $[\text{begin}_i, \text{end}_i)$。同一时间只能有一个活动，求最多能安排多少个活动。

- 按结束时间升序排序；依次扫描，若开始时间 $\geq$ 上一个结束时间，就选。
- 为什么不是开始早或时间短？开始早的活动可能很长；时间短的活动也可能卡在中间。
- 交换论证：设 g 结束最早，最优解第一个是 f；g 结束不晚于 f，把 f 换成 g 后，后面的活动仍相容，个数不变。

本页要点

区间不重叠问题的标准贪心：按右端点升序，能选就选。

活动选择 | 时间轴上的选择



- 选 $[1,4]$ 后，$[3,5]$ 冲突跳过；$[5,7]$ 可以从 5 开始，选。
- 同理选 $[8,11]$、$[12,14]$，共 4 个，与样例答案一致。

本页要点

判断条件是 begin $\geq$ last_end，端点相接也算相容。

例 6.5 | 样例手推与完整排序

样例 11 个活动，按结束时间升序：

结束	4	5	6	7	8	9	10	11	12	13	14
开始	1	3	0	5	3	5	6	8	8	2	12
选?	✓			✓				✓			✓

1. 选 $[1,4]$，当前结束 $=4$。
2. $[3,5]$ 开始 $3<4$，跳过；$[0,6]$ 跳过；$[5,7]$ 开始 $5\geq 4$，选，结束 $=7$。
3. $[3,8],[5,9],[6,10]$ 都开始早于 7，跳过。
4. $[8,11]$ 开始 $8\geq 7$，选，结束 $=11$。
5. $[8,12],[2,13]$ 跳过；$[12,14]$ 开始 $12\geq 11$，选。

共 4 个。注意 $[1,4]$ 与 $[5,7]$ 之间空了 1 秒，不影响相容性。

本页要点

判断条件是 begin $\geq$ last_end，端点相接也算相容。

例 6.5 | 完整程序（上）：读入并按结束时间排序

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

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int n;
    cin >> n;
    vector<pair<int, int>> act(n); // {结束时间, 开始时间}
    for (int i = 0; i < n; ++i) {
        int b, e;
        cin >> b >> e;
        act[i] = {e, b};
    }
    sort(act.begin(), act.end()); // 按结束时间升序
```

例 6.5 | 完整程序（下）：扫描并统计

```
    int chosen = 0;
    int last_end = INT_MIN; // 上一个选中活动的结束时间
    for (auto& [e, b] : act) {
        if (b >= last_end) { // 端点相接也算相容
            ++chosen;
            last_end = e;
        }
    }
    cout << chosen << '\n';
    return 0;
}
```

- 上一页与本页合起来是一个完整程序。
- 排序后 pair 的第一分量是结束时间，扫描时用第二分量判断。

例 6.5 | 复杂度、易错点与变式

- 排序 $O(n \log n)$，扫描 $O(n)$，总时间 $O(n \log n)$，空间 $O(n)$。
- 排序关键字是结束时间，不是开始时间，也不是持续时间。
- 若题目要求输出活动方案，需要把原始编号一起存下来；本题只要求输出个数。
- 区间是“开始等于上一个结束”可以接上；若题目规定不能相接，把 `b >= last_end` 改成 `b > last_end`。
- 若要求“最少几个礼堂容纳全部活动”，那是求最大重叠数，用扫描线或堆，不是本题的活动选择。

同类问题

“最多不重叠区间”“最少点覆盖所有区间”“区间完全覆盖”看起来都像区间贪心，但排序关键字和选择规则各不相同，不能混用。

本页要点

先明确优化目标，再决定按左端点还是右端点排序。

整数区间：用最少的点刺穿所有区间

给定 n 个闭区间 $[a_i, b_i]$，选一个元素个数最少的整数集合，使每个区间至少包含集合中的一个整数。

- 按右端点 b_i 从小到大排序。
- 维护最后选中的点 `last`。若当前区间满足 $a_i > \texttt{last}$，说明它没有被覆盖，就在它的右端点 b_i 放一个新点。
- 为什么选右端点？点越靠右，越有机会同时覆盖后面那些区间。

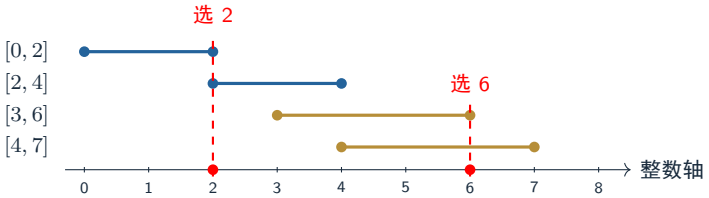
交换论证

对第一个尚未覆盖的区间，任何可行解都必须在其中放一个点。把这个点换成该区间的右端点，它仍然覆盖这个区间，并且不会减少对后面区间（右端点更靠右）的覆盖能力。

本页要点

按右端点排序；未被覆盖就选右端点，点尽量靠右。

例 6.6 | 四个区间与两个点



- 排序后：$[0,2],[2,4],[3,6],[4,7]$。
- 选 2：覆盖前两个区间（$[2,4]$ 含端点 2）。
- $[3,6]$ 的左端点 $3>2$，未被覆盖，选右端点 6。
- $[4,7]$ 含 6，已被覆盖。共 2 个点，与样例一致。

本页要点

闭区间端点相等算覆盖；若区间是开区间，判断要改成严格不等号。

例 6.6 | 完整程序（上）：读入并按右端点排序

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

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int n;
    cin >> n;
    vector<pair<int, int>> seg(n); // {右端点, 左端点}
    for (int i = 0; i < n; ++i) {
        int a, b;
        cin >> a >> b;
        seg[i] = {b, a};
    }
    sort(seg.begin(), seg.end()); // 按右端点升序
```

例 6.6 | 完整程序（下）：未被覆盖就选右端点

```
    int chosen = 0;
    int last = -1; // 坐标非负，-1 表示还没选点
    for (auto& [b, a] : seg) {
        if (a > last) { // 当前区间未被覆盖
            ++chosen;
            last = b; // 在右端点放一个新点
        }
    }
    cout << chosen << '\n';
    return 0;
}
```

- 上一页与本页合起来是一个完整程序。
- `pair` 的第一个分量是右端点，所以 `sort` 后按右端点升序。
- `last` 是最后选中的点；若当前区间左端点大于它，就必须再选一个。

例 6.6 | 复杂度与易错点

- 排序 $O(n \log n)$，扫描 $O(n)$，空间 $O(n)$。
- 本题坐标 $0 \le a \le b \le 10000$，所以 `last` 初值取 -1 即可；若坐标可能为负，初值要取负无穷。
- 必须按右端点排序。按左端点排序时，选右端点不再保证最优。
- 只统计点的个数，不需要输出具体集合；若题目要求输出方案，每次选点时把 b 存进数组即可。
- 同一个点可以覆盖多个区间，闭区间端点相等也算覆盖。

问题	排序关键字	每步选择
最多不重叠区间	右端点升序	能接上就选
最少点覆盖所有区间	右端点升序	未覆盖就选右端点
最多重叠区间数	左端点扫描	进入加一，离开减一

本页要点

区间贪心的核心不是“区间”，而是“你要优化的量”。

写贪心前先回答四个问题

1. 我每一步选择的具体对象是什么？是一个人、一个活动、一个数字，还是一条边界？
2. 我判断“当前最好”的关键字是什么？为什么不是它的反面（更早、更短、更大）？
3. 有没有反例？如果没有，能否用交换论证说明“任意最优解都能改成贪心解”？
4. 相等、边界、无解怎么处理？时间相同、区间端点相接、$n = 1$、全部相等，都要单独想一遍。

写代码前的三十秒

把这四个问题的答案写在草稿纸上。代码只是把答案翻译成排序和循环；答案错了，代码再漂亮也会 WA。

本页要点

贪心题的失分，绝大多数不是语法错误，而是排序关键字或相等处理错误。

常见错误清单

- 不证明就提交：样例过不代表策略对，先造小数据找反例。
- 排序关键字错：活动选择按开始时间、整数区间按左端点、分数背包按总价值，都是典型错误。
- 相等处理错：活动结束相同、接水时间相同、导弹高度相同，都可能影响答案或输出顺序。
- 溢出：等待时间、总高度、过河时间用 `int` 可能不够；先估算最大量级，再选 `long long`。
- 多组数据：没有读到文件结束，或上一组的状态没有清空。
- 输出格式：两位小数、保留一位小数、逗号分隔、行末不能多空格，都是判题的一部分。

本页要点

提交前用“最小、最大、全部相等、负数或零”四类数据各测一遍。

模板一：排序后扫描

```
// 按证明过的关键字排序，再线性扫描
sort(a.begin(), a.end(), [](const Item& x, const Item& y) {
    return x.key < y.key;
});
long long ans = 0;
int last = INT_MIN; // 上一个选择留下的边界
for (const Item& x : a) {
    if (x.begin >= last) {
        ++ans;
        last = x.end;
    }
}
cout << ans << '\n';
```

- 适用：活动选择、整数区间；`last` 记录上一个选择的边界。

模板二：小根堆维护当前最优

```
// 小根堆：堆顶是最早空出来的资源
priority_queue<long long, vector<long long>, greater<long long>> pq;
for (int i = 0; i < m && i < n; ++i) {
    pq.push(w[i]);
}
for (int i = m; i < n; ++i) {
    long long t = pq.top(); // 最早空出的水龙头
    pq.pop();
    pq.push(t + w[i]);
}
long long ans = 0;
while (!pq.empty()) {
    ans = max(ans, pq.top());
    pq.pop();
}
cout << ans << '\n';
```

- 适用：接水问题、合并果子；堆顶是当前最早空出的资源。

模板三：单调栈删除或保留

```
// 删掉 k 个数字，使剩下的数最小
string st;
for (char c : s) {
    while (k > 0 && !st.empty() && st.back() > c) {
        st.pop_back();
        --k;
    }
    st.push_back(c);
}
while (k > 0 && !st.empty()) { // 没有下降点，从末尾删
    st.pop_back();
    --k;
}
```

- 适用：删数、最小新整数；栈内始终保持不下降。

本章贪心策略与复杂度总表

题目	贪心关键字	数据结构	复杂度
排队接水	接水时间升序	排序	$O(n \log n)$
均分纸牌	从左到右结清差额	扫描	$O(n)$
删数问题	删除第一个下降点	单调栈	$O(\|n\|)$
拦截导弹	最合适系统 / LIS	排序或 DP	$O(n^2)$
活动选择	结束时间升序	排序	$O(n \log n)$
整数区间	右端点升序	排序	$O(n \log n)$

- 排序类题目，时间主要花在排序；扫描本身是线性的。
- 单调栈和堆的复杂度都来自“每个元素最多进出一次”。
- 写复杂度时要把输入规模代进去，例如 $n \le 1000$ 时 $O(n^2)$ 可行，$n \le 10^5$ 时就必须优化。

本页要点

先判断数据范围能接受哪种复杂度，再决定要不要写更难的贪心。

考场检查清单

1. 读题：输入有没有多组？是否读到文件结束？有没有“0 结束”的哨兵？
2. 建模：每一步的选择是什么？目标函数是什么？
3. 贪心：排序关键字是什么？相等时怎么办？能否举出反例？
4. 实现：下标从 0 还是 1？边界取不取等？循环范围对不对？
5. 数据：最小值、最大值、全部相等、单元素、负数或零，是否都试过？
6. 输出：小数位数、空格、换行、分隔符，是否与题面完全一致？

调试建议

先写一个暴力程序，对小数据随机对拍；贪心题最怕“样例对了，但策略只在样例上对”。

本页要点

能对拍就不要只靠眼睛；能证明就不要只靠对拍。

练习地图：先做哪几道？

- 排序贪心：金银岛（单位价值）、书架（高度降序）、电池的寿命（先判断最大电池）、寻找平面上的极大点（按 x 降序扫描）。
- 调度与模拟：接水问题（小根堆）、Crossing River（两种过河方案取小）、Ride to Office（选最早到达的人）。
- 数字与位运算：An Easy Problem（二进制中 1 的个数不变）、最小新整数（删数问题的多组版本）。
- 综合：装箱问题（分类讨论）、最大子矩阵（压缩列 $+$ 最大子段和）。

建议顺序

先做“书架、电池的寿命、接水问题”，再做“金银岛、最小新整数、Crossing River”，最后挑战“装箱问题、An Easy Problem、最大子矩阵”。

本页要点

练习不是重复例题：每道题都换了一个排序关键字或一个维护结构。

本章练习题索引 | 1–6 / 11

1. P5228 | An Easy Problem（一本通 1223）
2. P5229 | 最大子矩阵（一本通 1224）
3. P5230 | 金银岛（一本通 1225）
4. P5231 | 装箱问题（一本通 1226）
5. P5232 | Ride to Office（一本通 1227）
6. P5233 | 书架（一本通 1228）

点击蓝色题名进入学校 OJ；本章截图中的练习全部列在索引中。

本章练习题索引 | 7–11 / 11

7. P5234 | 电池的寿命（一本通 1229）
8. P5235 | 寻找平面上的极大点（一本通 1230）
9. P5236 | 最小新整数（一本通 1231）
10. P5237 | Crossing River（一本通 1232）
11. P5238 | 接水问题（一本通 1233）

点击蓝色题名进入学校 OJ；本章截图中的练习全部列在索引中。