贪心-基本操作

参考 OI-wiki 上的总结： https://oi-wiki.org/basic/greedy/

贪心算法（greedy algorithm），是用计算机来模拟一个“贪心”的人做出决策的过程。这个人十分贪婪，每一步行动总是按某种指标选取最优的操作。而且他目光短浅，总是只看眼前，并不考 虑以后可能造成的影响。 可想而知，并不是所有的时候贪心法都能获得最优解，所以一般使用贪心法的时候，都要确保自 己能证明其正确性。

贪心思想

1. 相邻交换法：如果交换方案中任意两个元素/相邻的两个元素后，答案不会变得更好，那么可以推定目前的解已经是最优解了。
2. 最优解性质：对于最优性问题，一个好的思路是 **逆向思维**，思考最优解满足何种性质（即 最优解的必要条件，例如单调性、连续性、不交叉等等），由这些性质切入往往可以简化 题目。
3. cut&paste：构造一个最优解，将当前贪心得到的这一部分解“复制粘贴”到最优解中，若最优解不会变差，则可证明贪心不会丢失最优解；换句话说，肯定有某些最优解包含当前贪心得到的这一部分解。例如MST、哈夫曼编码。
4. 最优性&可行性：若能同时说明某个解的 **最优性&可行性**，则能够说明其就是最优解。
5. 归纳法：先算得出边界情况（例如 $n=1$ ）的最优解 ，然后再证明：对于每个 F_{n+1} 都可以由 F_n 推导出结果。
6. 证明在任意局面下，做出局部最优决策之后，在可达解集中包含了做出其他任何决策后的可达解集（不会丢失任意解），即这个局部最优策略提供的可能性包含其他所有策略提供的可能性。
7. 在设计贪心算法时，手玩样例的能力往往至关重要。
8. 能贪心的问题必须要具有 **最优子结构**，这一点和DP相同。所不同的是DP需要计算所有子问题的解然后进行比较、选择其中最优的，而贪心则是**按照某种准则直接进行决策**。这提示我们对于有决策的题目，都可以首先设计DP解法，发现复杂度过高且不能优化之后，尝试贪心解法。

取当前最优

U130967 冰红茶(tea)

因为每种冰红茶越往后买单价越高，所以可以用小根堆维护当前每种冰红茶的单价，每次取最便宜的，再把该种冰红茶的单价更新后放回小根堆

时间复杂度 $\mathcal{O}((n+m)\log m)$

P1631 序列合并

方法 1：

设 $C[i][j]=A[i]+B[j]$，可以发现 C 数组每行内部单调不降，把每一行看成一类物品，就转化成了冰红茶模型，不需要把 C 数组造出来，只需要把 $C[i][1]$ 先放进小根堆里，并记录第二维的下标，每次取出后把下标往后推一位，更新 C 的值再放进去即可

时间复杂度 $\mathcal{O}(n\log n)$

方法 2：

还是考虑 $C[i][j]=A[i]+B[j]$，可以发现当 $1\le x\le i,1\le y\le j$ 时，$C[x][y]\le C[i][j]$，因此可以认为 $C[i][j]$ 只可能排在 C 数组排序后的第 $i\times j$ 位或更靠后的位置，当 $i\times j>n$ 时我们就不用考虑这个数了，因此我们只需要考虑 $i\times j\le n$ 的情况，取出这总共 $\mathcal{O}(n\log n)$ 个数，排序取前 n 为即可，先 `nth_element` 再排前 n 位可以让复杂度控制在一个 $\log$

方法 3：

可以二分找到第 k 小的和是多少，也就是二分答案统计每个 $A[i]$ 搭配多少个 $B[j]$ 能满足 $A[i]+B[j]\le mid$ 加起来（可以二分计算），判断其是否 $\ge k$

设第 k 小的和是 x，我们可以先把 $A[i]+B[j]<x$ 的数取出来放到一个数组里排序输出，再补上若干个 x 使得输出的总数够 k

注意不能直接把 $=x$ 的都拿出来，因为算上 $=x$ 的数以后总量可能 $>n$，甚至可能会把 n^2 个数都拿出来，这就保证不了时间复杂度了

时间复杂度 $\mathcal{O}(n\log n\log A)$

CF912D Fishes

把鱼放在 (x,y) 时对答案的贡献是好求的，就是 $(\min(n,x+r-1)-\max(x,r)+1)\times(\min(m,y+r-1)-\max(y,r)+1)$，计算方法就是考虑网的右下角所在的范围

显然把第一条鱼往中间放是最好的，后边的鱼与放在之前占过的格子往四周扩展一格的地方，这个过程可以用 bfs 实现，使用大根堆维护候选点，每次取出最优点，把周围四个点中没用过的放进堆里

时间复杂度 $\mathcal{O}(k\log k)$

P1315 [NOIP2011 提高组] 观光公交

等待时间总和等于每个人下车时间的总和减去其 T_i，因此我们只需要最小化每个人下车时间的总和

设每个车站最后上车的一个人的上车时间为 $last[i]$，先求出不用加速器时，车到每个车站的时间 $car[i]=\max(car[i-1],last[i-1])+d[i-1]$

设 $wait[i]=\max(0,\max(car[i]-last[i]))$，表示这一站的最后一个人等车的时间，如果在 i 处用加速器，就可以从 $i+1$ 开始找到连续一段 $wait[i]>0$ 的区间，设其被 r 卡停（$wait[r]$ 是 $=0$ 的），则在 $i+1\sim r$ 的 car 值都会减 1，在这些地方下车的人所用的时间都会减少 1

因此我们可以求出从每个点会被哪个点卡停（从后往前递），然后用前缀和计算出相应的这段区间有多少人下车，就能够计算出如果在这个点用一次加速器能使答案减小多少了。

每次取最优的决策点（注意不能用 d 已经为 0 的决策点）进行操作并重新计算 car 和 $wait$ 数组即可，时间复杂度 $\mathcal{O}(kn)$

可以看出其也符合冰红茶模型中，每次操作对答案的影响是越来越弱的

本题也存在 $\mathcal{O}(n\log n)$ 解法（加强版题目：LOJ 2601）

思想就是先考虑第一次操作，最好是能在 1 处用加速器，让在 $[2,n]$ 下车的人的时间都减 1，称这个操作是对区间 $[1,n]$ 操作，能用多少次呢，被三个信息限制，首先是总次数 k，其次是第一段路的长度 $d[1]$，再然后是 $wait[2]\sim wait[n-1]$ 里的最小值（如果是 0 就不可能让 $[2,n]$ 整个区间里的人的下车时间同时减 1，否则就可以），求出这个操作次数，从 k 里减掉，再把 $wait[2]\sim wait[n-1]$ 做区间修改（其实 $car[n]$ 也会减小，但 $wait[n]$ 不重要）如果 $k=0$ 了就结束了，如果 $d[1]=0$ 了，就去考虑区间 $[2,n]$，否则 $wait[i]$ 会有被减到 0 的点，记其为 p，就去考虑区间 $[1,p]$ 和 $[p,n]$，拆分出的区间要始终保持 $l\ne r$，如果拆分后的区间是 $l=r$ 的就不用考虑这个拆分后这个 $l=r$ 的区间了

设要考虑的区间为 $[l,r]$，则把区间按 $[l+1,r]$ 中下车的人数排序放入大根堆中，每次取出下车人数最多的区间操作然后把区间拆分即可，做一次操作会使一个 d 或一个 $wait$ 降到 0，因此总操作数是 $\mathcal{O}(n)$ 量级，操作一次需要对 $wait$ 做区间查询和区间修改（区间查询和修改都是 $[l+1,r-1]$ 区间，理论上 $car[r]$ 也会减小，但要么 $r=n$，要么当前的 $[l,r]$ 是一次拆分的左区间，此时 $wait[r]$ 一定已经是 0 了），所以需要对 $wait$ 使用线段树维护

总时间复杂度 $\mathcal{O}(n\log n)$

最优性和可行性

往往需要一定的观察力和经验。

U86308 01翻转(flip)

如果考虑区间翻转对原数组的影响是比较复杂的，但是区间翻转对差分数组的影响只是两个单点翻转，因此我们考虑在差分数组上操作

可以发现差分数组每一个不为 1 的地方都是需要调整的，而一次调整最多可以将 2 个 1 变为 0，不过我们不考虑最小次数，要考虑的是翻转的最短长度最长，那就每次只把一个 1 变成 0（相当于选个后缀）或者带上 $d[1]$ 一起翻转（选个前缀，最后如果是全 1 再整个翻回来）

所以最短长度最长值就是对 $d[2]\sim d[n]$ 里是 1 的地方对应的 $\max(i-1,n-i+1)$（选 $[1,i-1]$ 这个前缀或者 $[i,n]$ 这个后缀）取 $\min$

P5019 [NOIP2018 提高组] 铺设道路

从前往后考虑，截止到 $i-1$ 的线段肯定有 $d[i-1]$ 条，如果 $d[i]\le d[i-1]$，可以从中选 $d[i-1]$ 条延申过来，不用新增线段，否则必须新增 $d[i]-d[i-1]$ 条线段

拓展：如果每次可以使深度减任意正整数但不能减到负，且任意两次操作的线段要么是包含关系要么是相离关系，求最少次数

- 方法 1：栈，如果当前高度小于栈顶，就不断弹栈，弹栈后如果栈顶等于当前高度就不用新增操作，否则就新增一次操作并把当前高度入栈，栈中维护的就是操作方案，因为线段只能是包含或相离关系，所以后进先出，时间复杂度 $\mathcal{O}(n)$
- 方法 2：分治，每次找到当前区间的最低点，如果比之前的高就做一次操作把所有数加到这个高度，然后用最低点把区间拆成左右两边，因为要求区间最值，时间复杂度 $\mathcal{O}(n\log n)$，如果用笛卡尔树可以做到 $\mathcal{O}(n)$

P9868 [NOIP2023] 词典

对第 i 个字符串来讲，只需要判断其内部字符按字典序正序排是否比其它字符串内部字符按字典序倒序排更小即可

硬做的话可以每个字符串内部从大到小排，做前后缀 $\min$，然后用当前字符串内部字典序正排去跟做出的前后缀 $\min$ 比较，时间复杂度 $\mathcal{O}(nm\log m)$

因为字符串两两不同，可以自己跟自己也比一下，这样就是跟全局 $\min$ 比了，因为自己内部字符都相同时有自己正排等于倒排，所以我们判断的时候相等也算成功

更进一步，可以发现，如果当前字符串的最小字符大于其它字符串的最大字符，肯定就失败了，如果等于，则其它字符串倒序排后，后边字符小于等于开头，自己正序排，后边字符大于等于开头，又不会比不出来（比不出来是两个字符串内部都是同一字符的情况，此时两字符串相等），所以这样也会失败

因此我们只需要求出每个字符串的最小最大字符，求每个字符串的答案时，枚举一下其它串，判断一下即可

时间复杂度 $\mathcal{O}(nm)$，理论上这个也可以继续前后缀 min 优化之类的但没必要了

排序与相邻交换法

P1080 [NOIP2012 提高组] 国王游戏

设第 i 个人左右手上的数分别为 a_i, b_i，考虑交换第 i 个人和第 $i+1$ 个人的位置，设前 $i-1$ 个人的人左手上的数的乘积为 s

则需要找到合适的顺序使 $\max(\lfloor \frac{s}{b_i} \rfloor, \lfloor \frac{s \times a_i}{b_{i+1}} \rfloor) \leq \max(\lfloor \frac{s}{b_{i+1}} \rfloor, \lfloor \frac{s \times a_i}{b_i} \rfloor)$

显然如果找到一个排序方法满足把下取整都去掉以后的 $\leq$，那么也会满足带取整的 $\leq$，所以我们在之后的推导中去掉这个取整

两边同时除掉 s 再同时乘 $b_i \times b_{i+1}$，得到 $\max(b_{i+1}, a_i \times b_i) \leq \max(b_i, a_{i+1} \times b_{i+1})$

这个式子是不好证传递性的，我们要找到一个有传递性的式子满足只要按这个式子的顺序排，也能符合原式的排序条件，也就是说找到的式子可能是等价于原式或要求更严格的

分类讨论：

当左侧 $\max$ 取在 b_{i+1} 上时，不等式一定会成立，当左侧 $\max$ 取在 $a_i \times b_i$ 上时，只要 $a_i \times b_i \leq a_{i+1} \times b_{i+1}$ 不等式就一定成立

最后需要注意 $sort$ 的比较函数里相等的必须返回 $false$，所以比较条件为 $a_i \times b_i < a_{i+1} \times b_{i+1}$

这个分类讨论对应成更严谨的形式化推导就是把 $\max$ 拆成逻辑表达式（忽略等号）：

$((b_{i+1} < b_i) \wedge (a_i \times b_i < b_i)) \vee ((b_{i+1} < a_{i+1} \times b_{i+1}) \wedge (a_i \times b_i < a_{i+1} \times b_{i+1}))$

然后发现 $a_i \times b_i < a_{i+1} \times b_{i+1}$ 为 $true$ 就能让该不等式结果一定为 $true$

如果不忽略等号，只要按 $a_i \times b_i \leq a_{i+1} \times b_{i+1}$ 也能保证不等式成立

P2123 皇后游戏

相邻交换法 - 洛谷专栏 (luogu.com)

CF1043E Train Hard, Win Easy

考虑一对选手 (u, v) 做出 u 选第一类题目 v 选第二类题目这个决定的条件，条件是 $x_u + y_v < y_u + x_v$，移项得 $x_u - y_u < x_v - y_v$，如果我们按 $x - y$ 对人排序，会发现 i 在跟 $[1, i-1]$ 配对时会选择第二类问题，在跟 $[i+1, n]$ 配对时会选择第一类问题，做一些前缀和就可以算出每个人参加比赛扣的总分，再单独考虑不能组队的 m 个限制，把这些比赛场次所扣的分数从答案中减去即可

序列匹配

对两个数组进行匹配使某个值最优（如：火柴排队，即排序不等式）或在一个数组中每次选择两个数配对使某个值最优

P5021 [NOIP2018 提高组] 赛道修建

二分答案后求最多能找到多少条互不重叠且长度均 $\geq mid$ 的路径

对于子树 u，只有最多一条链能经过 u 走到 u 的父亲，因此如果我们能在子树内匹配路径就没必要把它延伸到父亲，因为在子树内匹配已经能给答案贡献 1 了，没必要占掉这个宝贵的上传名额，把暂时配不了的传上去说不定还能在上边接着匹配

我们子节点 v 传给父节点 u 的路径长度（算上 $u \to v$ 这一段后）为 len_v，先去把本身已经够 mid 的路径算进答案，然后就是要给这一堆 $< mid$ 的 len 做两两匹配，使尽量多对满足 len 之和 $\geq mid$，并且未匹配的最大值尽量大

贪心方法为：从小到大看每条路径，每次选能与之匹配的长度最小的进行匹配

方法 1：可以直接用 `multiset` 存下所有待匹配的 len，每次取 `begin` 并 `lower_bound` 找到可以匹配的最优路径，把这一对从 `multiset` 里删除，如果没法匹配就只删 `begin`

方法 2：将 len 从小到大排序，从小到大看每条路径，用一个指针从后往前扫，把能配上当前路径的路径都放到栈里，这个栈从栈顶到栈底 len 依次递增，每次是新入栈一些路径后取栈顶匹配当前路径

方法 3：将 len 从小到大排序，先假设不上传路径，求出最多能匹配多少对，用双指针，l 从左往右，r 从右往左，如果 $len_l + len_r \geq mid$，就把 l, r 匹配，`l++; r--;` 否则没有能匹配上 l 的，就 `l++`，这样做正确性依赖于能和当前 l 配的无论哪个都能和之后的 l 配，随便取一个（哪怕像这里取最大的）都没问题，但这个没考虑未匹配的最大值尽量大，可以二分上传路径的长度，把相应路径上传，然后再做这个匹配看匹配数是否会减少来决定把上传路径的长度调小还是调大

时间复杂度均为 $\mathcal{O}((n+m)\log n \log(suml))$

P1084 [NOIP2012 提高组] 疫情控制

首先二分答案，验证用 mid 时间是否可以成功

每个军队先尽力往上走（倍增跳），如果走不到根，就直接停在最高处，在这个点上打标记表示该子树已被控制，把能走到根的军队都存进一个数组

然后 `dfs` 一遍，对于 u，如果其所有子树 v 都已被控制，u 子树就整个被控制了，遍历根节点的儿子，如果其子树没被控制，把其存进一个数组

接下来就是分配哪个军队管哪个子树了，把军队按到根以后还能走多久从小到大排序，把需要控制的节点按根到这个节点的边长从小到大排序

按到根后的剩余时间从小到大看每个军队，它能控制的节点，后边的军队都能控制，所以可以任选一个控制，时间复杂度最低（不带 $\log$）的选法是双指针找到能控制且还未被控制的边长最小的节点，特殊地，如果该军队本身的子树还未被控制就不推双指针，直接让它留在自己这个节点，它能管的别人都能管，直接管自己就行，管自己不花时间也许别人还管不了，所以是对后续最好的结果

另一种处理留在自己这个节点的方法是看每个需要被控制的节点上有没有军队，如果有且走到根后剩余时间最少的军队走到根就回不到原来的点就让它留在这里，因为如果它去管别的距离根更近的点，别的军队就要过来管它，还不如直接让别的军队过去，也就是它能管的别人也能管，干脆管自己，管自己不花时间也许别人还管不了，做完这个处理再做双指针匹配。

时间复杂度 $\mathcal{O}(n \log n \log w)$

反悔贪心

无论当前的选项是否最优都接受，然后进行比较，如果选择之后不是最优了，则反悔，舍弃掉这 个选项；否则，正式接受。如此往复。 反悔特性往往由 **优先队列（或者单调队列）**来支持。

P3545 [POI2012] HUR-Warehouse Store

从前往后看，如果能卖则卖，不能卖则考虑能否反悔，如果之前有 b 更大的顾客被满足了，就去掉其中 b 最大的，换成当前顾客，可以把满足了的顾客的购买量用大根堆存储，方便反悔替换

P2209 [USACO13OPEN] Fuel Economy S

把油箱中的油想象成离散、一份一份的，不同价格的油不会混在一起。接着假设之前加的油，可以在之后的加油站“退货”（前提是不能消耗掉）

这样就可以贪心：

1. 路程中需要消耗燃油时，先消耗低价油；
2. 到加油站 i 时，将 $c[j] > c[i]$ 的高价油退货，加满 $c[i]$ 油

使用单调队列维护当前油箱的情况即可

无解的情况：

1. 相邻油站的间距（包括最后一个油站到终点的距离）大于油箱容量 M
2. 第一个加油站距离起点的距离 $>$ 初始油量 B

还有另一种直接贪心的做法：

首先记录当前位置和当前油箱里的油量，然后开始跑

- 如果加满油的情况下能跑到比当前油站便宜的油站，就跑到第一个比当前油站便宜的油站，如果当前油够就直接去，否则加恰好够用的油过去，这里可以把终点看成一个免费油站，这样就可以算上最后走到终点的操作了
- 如果加满油的情况下不能跑到比当前油站便宜的油站，就加满油，跑到能跑范围内最便宜的油站，也可以加满油先跑到下一站，因为如果下一站就是范围内的最便宜的油站，那么显然没问题，如果不是，那么就会进入第一种情况走到范围内最便宜的油站

这个做法需要求后继点和区间最值，如果第二种情况一步一步走，每走一步判一下就不用区间最值了

可以发现该策略形成的方案与反悔贪心形成的方案是一样的，第一条相当于剩余还没用的油从单调队列队尾弹出，第二条加的油会被完全消耗，而能跑范围内最便宜的油站之前的油都会被最便宜的油站弹出，就没必要去了

P4053 [JSOI2007] 建筑抢修

该类型题目的三种常用方法

反悔贪心：按时间从前往后模拟，使用数据结构存放已经选择的元素集合，若当前元素选不了则进行反悔；利用决策包含性（不损失决策）证明正确性

时光倒流：按时间从后前模拟，使用数据结构存放当前能选择的元素，贪心选当前最优的元素

贪心覆盖：按收益从大到小决策，每次尽量往后放

注意这个问题具有时间上的最优子结构，可以从前 t 天的最优解推出前 $1 \sim t-1$ 天的最优解

方法 1：贪心反悔

按 $t2$ 从小到大排序，用大根堆维护现在决定修的建筑的 $t1$，每次如果能修则修，否则如果 $t1$ 比堆顶小就反悔堆顶那个建筑加入当前建筑

方法 2：时光倒流

按 $t2$ 从大到小排序，每次新产生一个建筑，然后新获得了一段时间，用小根堆维护现在每个建筑还需要多少时间修完，先把新建筑放进去，然后弹堆（修建筑）直到堆为空或时间不足以修完堆顶建筑，此时堆顶的建筑可以先修剩余时间这么多，切记这一段的时间不能带入下一段，因为下一个新建筑不能用这段的时间，不用担心最终方案修建筑的时间不连续，总能调成连续的（把交叉的几段中结尾靠后的放后边，结尾靠前的放前边）

方法 3：贪心覆盖

按 $t1$ 从小到大排序，每次询问 $t2$ 及之前空余时间是否够 $t1$（不用要求连续，最终方案可以调整），然后把最后 $t1$ 长度的空位（一个后缀）覆盖，可以使用线段树支持这些操作

通过单调性将优先队列换为队列

如 01BFS 将 Dijkstra 的优先队列换为双端队列，只有 k 边种权的最短路可以用 k 个队列（队列内部有单调性）代替优先队列

P2827 [NOIP2016 提高组] 蚯蚓

首先有优先队列做法，用小根堆维护蚯蚓的长度，整体加 q 通过定义一个变量当作优先队列里的值和真实值的偏移量实现，注意切的时候从堆中取出蚯蚓后要先加偏移量再乘比例再把偏移量减了放回堆

然后观察出一个性质：先切的蚯蚓切出的两段的长度一定分别不小于后切的蚯蚓切出的两段的长度

简单证明：设 $a \geq b$，在该切 a 的时候 b 长度为 b，a 在 b 前边 t 秒切，该切 b 的时候，b 已经长到了 $b+tq$

则 a 切出的两段在 b 切完时长度为 $\lfloor pa \rfloor + tq, (a - \lfloor pa \rfloor) + tq$，$b$ 切出的两段长度为 $\lfloor p(b+tq) \rfloor, (b+tq) - \lfloor p(b+tq) \rfloor$

因为 $pa \geq pb$，所以 $\lfloor pa \rfloor + tq = \lfloor pa + tq \rfloor \geq \lfloor p(b+tq) \rfloor$

而 $a - \lfloor pa \rfloor$ 和 $b - \lfloor pb \rfloor$ 可以看成 $\lceil (1-p)a \rceil$ 和 $\lceil (1-p)b \rceil$，因此肯定有 $a - \lfloor pa \rfloor \geq b - \lfloor pb \rfloor \geq b - \lfloor pb + ptq \rfloor$

证明完毕

因此我们可以使用三个队列分别存储没切过的蚯蚓，切出的前半段，切出的后半段，每个队列内部都是单调不减的，每次取三者队首中最小的切

时间复杂度线性

P6033 [NOIP2004 提高组] 合并果子 加强版

显然越靠后的操作合并出的果子数越多，所以可以用两个队列维护没参与过合并的果子堆和合并出的果子堆，每次取两者队首最小的就是最小的果子堆，取两次然后把这两堆合并

为了避免最开始排序的 $\log$，本题 a 的范围为 10^5，可以使用计数排序