“卓见杯”2020 年河南省 CCPC 大学生程序设计竞赛题目解析

命题组

1 班委竞选

设 g_x 为第 x 类班干部最大票数。从小到大枚举学号 i，若 $t_i > g_{c_i}$ 则更新 $g_{c_i} \leftarrow t_i$ 并且记录 i 为 c_i 的答案。最后输出答案即可。

2 广告投放

使用数论分块优化 DP。需要掌握以下前置知识：

1. $\lfloor \lfloor n/i \rfloor /j \rfloor = \lfloor n/(i \cdot j) \rfloor$
2. $\lfloor n/i \rfloor$ 的取值只有 $\mathcal{O}(\sqrt{n})$ 种（数论分块、整数分块）

记 $dp_{i,c}$ 是放完前 i 集后还有 c 名观众的答案，有两种转移：

$$dp_{i,c} \to dp_{i+1,c}$$

$$dp_{i,c} + c \cdot p_i \to dp_{i+1,\lfloor c/d_i \rfloor}$$

第二维有用的实际只有 $\mathcal{O}(\sqrt{m})$ 种取值。实现时有很多种方法，可以不带 log 地做到时间复杂度 $\mathcal{O}(n\sqrt{m})$。

3 我得重新集结部队

本题是一个模拟题，按题意实现即可，复杂度为 $\mathcal{O}(n^2)$。需要注意以下几个点：

1. 出现多只与狂热者距离相同的异虫时，需要选取最早出现的那只。
2. 考虑狂热者距离时不能考虑已经死亡（离开战场）的异虫。
3. 距离平方的极限数据可达 4×10^{16}，使用 `double` 计算有可能会产生浮点精度误差。但是注意到距离之间只需要比较大小，因而可将所有距离平方，在长整型范围内进行运算，避免精度误差。

4　园艺大师

先考虑 n 很小的版本，使用容斥原理计算答案。不妨定义 < 是合法的，设 m 是 > 的数量。那么 2^m 枚举所有 > 号的位置是否满足大于的要求，计算所有方案并进行容斥即可解决该规模下的问题。

对于本题的数据范围，不妨假设有 m 个要求是 >，它们的位置分别是 $p_1, p_2, \ldots, p_m$（$1 \le p_i < n$），并令全集 $U = \{p_1, p_2, \ldots, p_m\}$。令 S 为 U 的一个子集，定义函数 $f(S)$ 为 S 中的要求不满足 >（即满足 $\le$）且 $U - S$ 的要求不作限制时的合法方案数，则所有灌木可以根据 $U - S$ 中的位置分割成很多段，每一段里的要求只会是 < 和 $\le$，段间则没有要求。

不妨假设某一段被分割出来的灌木有 len 株，里面有 cnt 个要求是 $\le$，则本段的方案数为 $\binom{h+cnt-1}{len-1}$，最终 $f(S)$ 的值为每一段的方案数直接相乘。由容斥原理，题目所求答案为

$$\sum_{S \subseteq U} (-1)^{|S|} f(S)$$

考虑使用 DP 优化上式的计算。设 g_i 表示 i 是某一段结尾的前提下，$1 \sim i$ 灌木的修剪方案数。接下来可以枚举下一段的长度 j，将该段对答案的贡献转移到 g_{i+j} 上，即：

$$(-1)^{\text{cnt}} \binom{h + \text{cnt} - 1}{j - 1} g_i \to g_{i+j}$$

其中 cnt 表示 $[i+1, i+j]$ 这一段中 > 的数量（被容斥为 $\le$）。还需要注意的是只有 > 左边的位置和 n 才可能成为一段的结尾。

最终答案为 $g(n)$，总时间复杂度为 $\mathcal{O}(n^2)$。

5　发通知

将 $a_i, b_i + 1$ 离散化，记 $\text{map}[x]$ 为 x 离散化后的值。对于每个同学 $(a_i, b_i + 1, w_i)$，拆成 $(\text{map}[a_i], w_i, 1)$, $(\text{map}[b_i + 1], w_i, -1)$ 的两个点，将 $2n$ 个点按第一关键字权值从小到大排序。

扫一遍排序后的点，每次将第一关键字重合的点一起处理，将新加入的点第二关键字权值与变量 sum 异或，并累加第三关键字权值至 num；若处理完当前点第一关键字重合的所有点时满足 $\text{num} \ge k$ 则用 sum 更新答案。其中 sum 和 num 开始时均为 0。

扫描完输出答案即可。时间复杂度 $\mathcal{O}(n\log n)$。

6 旅游胜地

二分答案 + 2-SAT。

如果答案是 d，对于原题的一条边 $e=(u,v)$，如果 u 选用某个权、v 选用某个权时，绝对差超过了 d，那么这种权重的组合就不能选，或者说 u 选用这个权、v 必选另一个权，v 选用这个权 u 必选另一个权。

点 u 的两个选权方法为 2-SAT 模型的点，上述的依赖关系为 2-SAT 模型的边。

如果当答案为 d_0 时，2-SAT 有可行解，那么这个解对于更大的答案显然都是可行的，所以能二分答案。

时间复杂度通常能做到 $\mathcal{O}\left((n+m)\log a\right)$。

7 vvvvvvvim

对比第一段文本和第二段文本。如果它们完全相同，那么可以将第一段文本的某个字符“修改”成该字符。否则，考虑所有字符不同的位置，如果这样的位置中，第二段文本出现了两种以上的字符，那么一定不可行，因为操作只允许修改成相同的字符。设需要改成的字符是 ch。仅仅考虑需要修改成 ch 的位置是不够的，我们可以在路径中加入一些本来就是 ch 的位置，以使得路径连通。例如 `bab` 修改成 `aaa`，就需要同时修改中间的位置以使得路径连通。

简而言之，我们需要找出所有第二段文本中 ch 的位置，并判断其中需要修改的位置是否连通。但是由于题目给出的文本规模很大，无法直接建图。这里定义一个区间为文本某行的一个连续段。注意到一个区间内可以相互到达，因而整体的连通性等价于区间的连通性。为此，需要先对两段文本的区间进行分割，使得两段文本的区间完全相同。例如第一段文本是 $[1,3],[4,6]$，第二段文本是 $[1,2],[3,6]$，那么应当将两段文本都分割成 $[1,2],[3,3],[4,6]$。这里有多种实现方式，例如可以将区间的分界线归并起来，即可得到新的分割方法。

分割区间后，选取出那些字符为 ch 的区间进行建图。边有两种，一种为行内的边，一种为相邻行之间的边。行内的边较为简单，只有两个相邻区间之间才可连边。行间的边可以使用 `two pointer` 方法求出。考虑第 i 行

的第 j 个区间与第 $i+1$ 行的第 $[l_j, r_j]$ 个区间相交，那么显然 l_j 和 r_j 分别是单调增的。这种方法也同时可以证明边数是线性的。

时间复杂度 $\mathcal{O}(\sum d)$。

8 林克与翻转排列

为了方便讨论，我们先将 a 和 b 一起重标号，使得 $b = 1, 2, \cdots, n$。

如果 $n = k$，那么有解的排列只有两种。

如果 $n = k+1$，注意到每个操作都是自己的逆操作，因此连续使用同种操作是没有意义的。从而有意义的操作序列只有 $1, 2, 1, 2 \cdots$ 和 $2, 1, 2, 1, \cdots$，可以发现这两种序列都会在 $\mathcal{O}(n)$ 步后回到 $1, 2, \cdots, n$。

接下来是最重要的 $n = k+2$ 的情况。如果 $n \mod 4 = 2$，可以发现任何操作都不改变逆序数的奇偶性，因此具有奇逆序数的排列 a 无解。我们给出其它情况下的构造方法。注意到操作 $1, 2$ 使得排列变成 $a_{n-1}, a_{n-2}, a_1, \cdots, a_{n-3}, a_n$，又有 $n-1$ 是奇数，因此连续使用若干次 $1, 2$ 可以将 $a_1, \cdots, a_{n-1}$ 变成任意一种 `rotate`。$a_2, \cdots, a_n$ 同理。同时注意到，这种方法是两步两步进行操作的，不会改变逆序数的奇偶性。

我们归纳地进行构造。不妨设现在已经将 $1 \sim i (1 \le i \le n-3)$ 排在了正确的位置，假如此时 $i+1$ 在 n，那么可以先把 $1 \sim i$ 旋转到 $i+1$ 的左边，然后把 $1 \sim i+1$ 向左旋转一位，再旋转到串的开头。

否则的话，把 $i+1$ 旋转到 1 的位置。容易看出此时 $1 \sim i$ 仍是连续的，我们可以利用 $2, 3$ 将它们旋转到 $n-i \sim n-1$ 的位置，然后再利用 $1, 2$ 将 $1 \sim i+1$ 旋转到正确位置。

现在，排列要么已经正确了，要么是 $1, 2, \cdots, n, n-1$。对于后一种情况，如果 $n \mod 4 = 2$，那么就无解了。对 $n \mod 4 = 0$ 的情况，我们可以在最开始进行一次操作，使得逆序数为偶数。这样就可保证得到正确的排列。

$n > k+2$ 的情况可以简单地归约到 $n = k+2$ 的情况，这里就不赘述了。

9 太阳轰炸

记 d 为火焰飞弹落点与原点的距离，火焰飞弹击中虚空碎片的充要条件是 $d \le R_1 + r$，那么一枚碎片击中的概率为 $p = \min\left(1, \left(\frac{R_1+r}{R_2}\right)^2\right)$。同时可

以根据 a, h 计算出至少需要击中 $c = \lceil \frac{h}{a} \rceil$ 次。根据二项分布的知识可知，答案为 $\sum_{i=c}^{n} \binom{n}{i} p^i (1-p)^{n-i}$。时间复杂度 $\mathcal{O}(n)$。

10 二进制与、平方和

10.1 按位维护数据结构 $\mathcal{O}(n \log n \log a)$

很容易能观察到，一个元素的某一位如果被 AND 成 0，那这一位就不会变化了。所以如果能够快速地得知被修改的区间里有哪些元素会有变换，则可以用某个数据结构维护一下区间元素平方的和，对每个有变化的元素暴力地去修改即可。

考虑每一位分别维护一个并查集或某个数据结构，用来记录这一位有哪些位置上的元素依然是 1。

对于并查集，可以将相邻的、这一位上都是 0 的元素并在一起，并记录一下最大的下标（连续的多个 0 的最右边），相应的下一个位置就是这些元素右侧第一个 1。查询就是询问 l 位置上右侧的第一个 1，对这个元素进行修改、置 0，然后和两侧的 0 合并一下，接下来再从这个位置重复进行相应的操作。

如果使用 $\mathcal{O}(\log a)$ 个并查集来维护每一位连续的 0 的情况、用一个线段树来支持单点修改、区间查询元素平方和的操作，总体的复杂度是 $\mathcal{O}(n \log a \cdot \alpha(n) + n \log n \log a) = \mathcal{O}(n \log n \log a)$ （视 n 和 q 同阶）。

10.2 在线段树上递归 $\mathcal{O}(n(\log n + \log a))$

实际上观察出一个元素的某一维最多会变化一次的情况下，显而易见地可以考虑在线段树上维护” 区间 OR”、” 区间元素平方的和”。

对于修改，当我们在处理某个节点的时候，如果修改操作连维护的区间 OR 都改不掉，那说明整个区间里是没有会变化的值，可以直接不做了；如果走到一个被修改操作的区间包括了的节点（通常线段树操作会在这时打上标记后返回），但修改操作是会影响到区间 OR 的情况下，就需要递归下去继续处理。

很显然每个位置最多会变化 $\mathcal{O}(\log a)$ 次，每次都会影响到祖先上所有的点，修改操作时也是每个祖先节点都被操作了一次。所以乍一看复杂度还是 $\mathcal{O}(n \log n \log a)$。

10.3 线段树做法更严格的时间复杂度上界证明

考虑把线段树的结构复制 $\mathcal{O}(\log a)$ 个出来，记第 i 个线段树为 S_i。S_i 中存在有原来线段树结构中的节点 u 的复制点，当且仅当原线段树节点 u 记录的区间 OR 的第 i 位为 1。

由于一个节点区间 OR 第 i 位为 1，其父节点的也必然是 1，所以 S_i 会一直是一个连通的图（也就是树，也可能是空图）。

下面说明：

1. 修改时，额外的递归操作等同于在这些线段树上砍子树；2. 一开始复制出来的线段树节点总数量，是总的额外递归次数的上界。

每次修改操作，如果是走到一个通常线段树本该停止递归、现在需要递归进行修改操作的节点 u 时，说明区间里存在至少一个元素的某一位会被修改操作改掉，记某个被修改到的位为第 i 位。

那么被递归的 u 子树中的节点，所有区间 OR 第 i 位为 1 的节点都会被递归到，并且递归结束后第 i 位都变成了 0。这就相当于我们把 S_i 中，u 所对应的节点的整个子树给砍掉了，子树中被删掉的节点都可以对应到某次递归操作。

u 子树中，如果某个节点的区间 OR 没有被修改操作修改过，那就不会递归到它（或者说递归到的时候会停止继续递归，这里浪费的时间挂给它的父节点，让父节点花费的时间乘 3 即可）。

S_* 中根据定义被删掉的点，要么是某次额外递归操作导致的，要么是通常的线段树操作更新区间 OR 时导致的。并且所有的额外递归操作，都至少能对应到 S_* 中的一个被删掉的点。

所以，S_* 中根据定义被删掉的点的数量“大于或等于”通常线段树本该停止递归、现在需要递归所进行的总递归次数”。因此，除通常线段树需要进行的操作次数外，额外进行的递归总次数是不会超过最开始复制出来的节点总数量，也就是 $\mathcal{O}(n\log a)$。

加上通常的线段树操作的时间复杂度，算法的总时间复杂度为
$\mathcal{O}(n(\log n+\log a))$（视 n 和 q 同阶）。

11 子串翻转回文串

如果原串是回文串，那么翻转自身即可满足要求，因此只考虑非回文串的情况。

首先可以发现，初始串两端翻转后相同的部分一定是无用的，因为要么得一起翻转，要么就只翻转一侧，且翻转后必然还得和原来保持相同。如果后者情况是合法的，那么将翻转部分两侧翻转后相同的部分去掉，则翻转结果也保持不变，因此初始串两端翻转后相同的部分可以直接去除，不用考虑。

去除后，串的两端字符一定不同，因此一定会以其中的某端为端点进行唯一的一次翻转。由于可以作为另一端端点的位置只有 $\mathcal{O}(n)$ 个，因此用哈希或其他结构暴力判断这 $\mathcal{O}(n)$ 个区间翻转后是否是回文串即可。

单次时间复杂度 $\mathcal{O}(n)$。

12 送外卖

首先，用 floyd 算法预处理出公寓所有点对间最短路 D_{ij}，并计算出每个订单在任意时刻的收益 $g_{i,t}$。

然后 DP，记 $T = \max_{1\le i\le n} w_{i,d_i}$，状态 $f[s][t][j]$ 表示已完成订单状态 s（$s < 2^n$，第 x 位为 1 表示第 x 号订单已完成，否则未完成）、当前时刻 t（$t \le T$）、最后完成的订单编号 j ($1 \le j \le n$)，初始状态，$f[0][0][1] = 0$，其他均为 -1；从状态 $f[s][t][j]$ 处进行转移，枚举下一未完成目标订单位置 k，$f[s|2^{k-1}][t+D_{jk}][k] = \max(f[s|2^{k-1}][t+D_{jk}][k],\ f[s][t][j] + g_{k,t+D_{jk}})$，其中 $(s\&2^{j-1}) = 1$ 、$(s\&2^{k-1}) = 0$。

统计答案。可以发现，当 $t \ge T$ 时，所有订单收益均变为 $p_{i,\ d_i+1}$。因此我们不计算 $t > T$ 的状态，通过枚举 $f[s][T][j] + \sum_{1\le k\le n} p_{k,\ d_k+1}$ 更新答案，其中 $(s\&2^{j-1}) = 1$、$(s\&2^{k-1}) = 0$。最后再用 $f[2^n-1][t][j]$ 更新答案即可。

复杂度分析：考虑状态 $f[s][t][j]$ 有效性，合法的 j 满足 $(s\&2^{j-1}) = 1$，即 j 遍历了 2^n 状态中所有的 1；转移过程中满足 $(s\&2^{k-1}) = 0$，即 k 遍历了 2^k 状态中所有的 0。由于 2^n 个状态中 0 和 1 出现次数相同。故有效状态数为 $((2^n \times T \times n)/2$，实际转移数为 $((2^n \times T \times n)/2 \times n)/2$ 次。

整体时间复杂度 $\mathcal{O}(2^n T n^2/4)$。