《比赛》命题报告

柯绎思

题意简述

有 n 名选手，编号分别为 $1, 2, \cdots, n$。编号为 i 的选手的实力值为 i。所有选手分为红、蓝两队。

现在将所有选手排成一个环，每一对相邻且不属于同一队的选手会进行一场比赛，实力值较大的选手获胜，他所在的队伍的得分增加一。

然而蓝队的选手勾结了裁判，如果一场比赛中**红队选手**获胜，且他在蓝队选手的**顺时针方向**，则这场比赛**不计入得分**。

对于每个 $k = -n, \cdots, -1, 0, 1, \cdots, n$，求出有多少种将选手排列的方法，使得红队得分恰好比蓝队得分大 k。

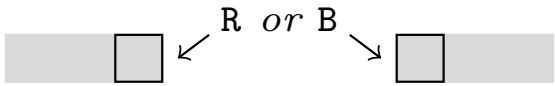
$3 \leq n \leq 3000$。

算法一：$\mathcal{O}\left(n^6\right)$

为了方便，下文把一个选手的逆时针方向称为左边，顺时针方向称为右边。

按照编号从小往大加入每个选手，维护所有选手在环上构成的连续段的信息。加入第 i 个选手时，他可能单独构成一个连续段，跟在某个连续段的左边或右边，或者把左右两个连续段连接成一个。对于每一对相邻的选手，在编号较大的人加入时统计上他们的贡献。

为了方便统计贡献，可以记录连续段之间的间隔的信息。对于一段间隔，考虑它左右两边第一个选手所属的队伍，共有四种情况。那么只需要分别记录符合这四种情况的间隔的数量即可。



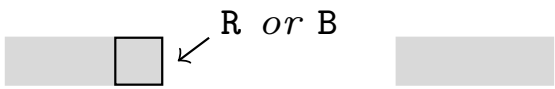
可以设计 $\mathrm{dp}(i, c_0, c_1, c_2, c_3, s)$ 表示当前加入了 $1 \sim i$ 号选手，四种情况的间隔的数量分别是 c_0, c_1, c_2, c_3，红队得分减去蓝队得分为 s 的方案数。容易做到 $\mathcal{O}(1)$ 转移，因此总复杂度为 $\mathcal{O}\left(n^6\right)$。

算法二：$\mathcal{O}\left(n^4\right)$

承接算法一的思路，考虑减少 dp 的状态数。先把每相邻两个人的贡献都加一（包括原本不进行比赛的）。也就是说如果本来红队加一分，则贡献 2；如果本来不比赛或不计入得分，则贡献 1；如果本来蓝队加一分，则贡献 0。

观察相邻两个人 p_i 和 p_{i+1} 的贡献，如果是 $p_i > p_{i+1}$ 即左边的人获胜，则此时不会出现“比赛不计入得分”的情况。可以发现贡献可以拆到两边：$[s_{p_i} = \mathtt{R}] + [s_{p_{i+1}} = \mathtt{B}]$。

这样可以稍微修改一下算法一中统计答案的方法。对于一对相邻的选手，如果左边的人获胜，那么可以在他们加入时，分别统计上拆到他们上面的贡献，这样就无需知道对方属于哪一队；如果右边的人获胜，还是在编号较大的人加入时统计上贡献。



可以设计 $\mathrm{dp}(i, c_0, c_1, s)$ 表示当前加入了 $1 \sim i$ 号选手，有 c_0 段间隔左边第一个选手属于红队，有 c_1 段间隔左边第一个选手属于蓝队，红队得分减去蓝队得分为 s 的方案数。容易做到 $\mathcal{O}(1)$ 转移，因此总复杂度为 $\mathcal{O}\left(n^4\right)$。

算法三：$\mathcal{O}\left(n^4\right)$

之前的算法似乎已经没有前途了，换一个思路，考虑容斥。还是先把每相邻两个人的贡献都加一。为了方便，使用生成函数的方式来描述贡献：如果原本蓝队得分加一，那么贡献就是 1；如果原本红队得分加一，那么贡献就是 x^2；其余贡献都是 x。

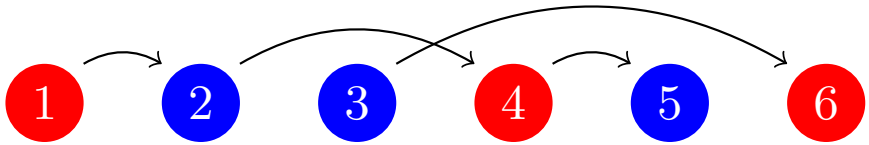
钦定一些顺时针方向的编号的上升段，考虑钦定后方案的贡献：

- 对于没有被钦定的地方，贡献的值就是左边的人获胜时的贡献。根据算法二，此时贡献可以拆开到两边的人上。也就是说，如果一条上升段的开头是 `B`，就贡献 x，否则贡献 1；如果一条上升段的结尾是 `R`，就贡献 x，否则贡献 1。
- 对于被钦定的地方，贡献的值就等于右边的人获胜时的贡献减去左边的人获胜时的贡献，也就是 `B` $\to$ `R` 的贡献是 $x-1$，`R` $\to$ `B` 的贡献是 $1-x^2$，其余贡献都是 0。

一个方案的贡献就是所有上述贡献的乘积。

不同上升段之间的相对顺序已经不影响贡献了，现在就把上升段看成从前往后的若干条链。

算法三：$\mathcal{O}\left(n^4\right)$



因此可以按照编号从小往大把选手插入若干条上升链。设 $\mathsf{dp}(i, c_0, c_1)$ 表示考虑了 $1 \sim i$ 号选手，有 c_0 条以 B 结尾的链，有 c_1 条以 R 结尾的链，所有方案的贡献之和是多少。

转移如下：

- 若 s_i 为 R：
  - $\mathsf{dp}(i, c_0, c_1 + 1) \leftarrow \mathsf{dp}(i-1, c_0, c_1)$
  - $\mathsf{dp}(i, c_0 - 1, c_1 + 1) \leftarrow \mathsf{dp}(i-1, c_0, c_1) \cdot c_0(x-1)$
- 若 s_i 为 B：
  - $\mathsf{dp}(i, c_0 + 1, c_1) \leftarrow \mathsf{dp}(i-1, c_0, c_1) \cdot x$
  - $\mathsf{dp}(i, c_0 + 1, c_1 - 1) \leftarrow \mathsf{dp}(i-1, c_0, c_1) \cdot c_1(1 - x^2)$

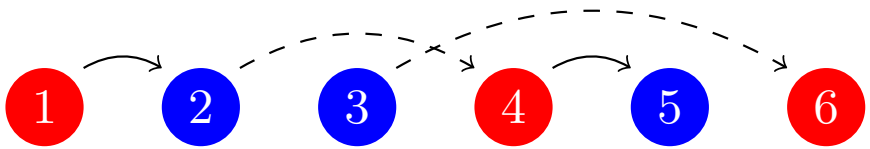
最后把所有链排成一个环，还要乘上 $(\text{链数} - 1)!$，且每条以 R 结尾的链再贡献 x，即答案的生成函数是 $\sum_i \sum_j (i+j-1)! \cdot \mathsf{dp}(n, i, j) x^j$。总复杂度还是 $\mathcal{O}\left(n^4\right)$。

算法四：$\mathcal{O}\left(n^3\right)$

承接算法三的思路，注意到算法三中 dp 的转移非常特殊，如果按照编号从小往大看，每一条有贡献的链都是 RB 交替的。

对于若干条 RB 交替的链，如果只提取出其中 B $\to$ R 的边，那么这些边构成一个匹配，其中每条匹配边都是先蓝后红，称这种匹配为 I 型匹配。同理，如果只提取出其中 R $\to$ B 的边，那么这些边也构成一个匹配，其中每条匹配边都是先红后蓝，称这种匹配为 II 型匹配。

反过来，任给一个 I 型匹配 M_1 和一个 II 型匹配 M_2，都能对应成若干条 RB 交替的链 $M_1 \cup M_2$。同时贡献关于 M_1、M_2 是独立的：M_1 中的边贡献 $x-1$，M_2 中的边贡献 $1-x^2$；一个 B 在链开头当且仅当它不在 M_2 中，一个 R 在链结尾也当且仅当它不在 M_2 中，因此每个不在 M_2 中的选手再贡献 x。



算法四：$\mathcal{O}\left(n^3\right)$

可以设 $F_{i,j}$ 表示考虑了 $1\sim i$ 号选手，有 j 个蓝队选手还未匹配，所有 I 型匹配的贡献之和是多少。设 $G_{i,k}$ 表示考虑了 $1\sim i$ 号选手，有 k 个红队选手还未匹配，所有 II 型匹配的贡献之和是多少。

转移还是容易的：

- 若 s_i 为 R，则 $F_{i,j}=F_{i-1,j}+(j+1)(x-1)F_{i-1,j+1}$，$G_{i,j}=G_{i-1,j-1}$。
- 若 s_i 为 B，则 $F_{i,j}=F_{i-1,j-1}$，$G_{i,j}=xG_{i-1,j}+(j+1)(1-x^2)G_{i-1,j+1}$。

其实这里 F,G 和算法三中的 dp 数组满足 $\mathrm{dp}(i,j,k)=F_{i,j}\times G_{i,k}$，如果直接观察 dp 转移式也可以得到这个结论。

最终答案就是 $\sum_i\sum_j(i+j-1)!F_{n,i}G_{n,j}x^j$。如果直接计算 n^2 次乘法还是 $\mathcal{O}\left(n^4\right)$ 的，但是可以先计算每个 $F_{n,i}$ 和 $G_{n,j}$ 的点值，最后再进行插值，这样总复杂度就是 $\mathcal{O}\left(n^3\right)$。

算法五：$\mathcal{O}\left(n^2 \log n\right)$

算法四显然做麻烦了，其实对于一个大小为 i 的 I 型匹配 M_1 和一个大小为 j 的 II 型匹配 M_2，$M_1 \cup M_2$ 的贡献是可以直接计算的，就是 $(x-1)^i \left(1-x^2\right)^j x^{n-2j}$。

不用在 dp 的过程中就维护多项式，可以直接先计算 a_i 表示大小为 i 的 I 型匹配的数量，b_j 表示大小为 j 的 II 型匹配数量。$\{a_i\}$ 和 $\{b_j\}$ 都可以在 $\mathcal{O}\left(n^2\right)$ 的时间复杂度内计算。

接下来，答案就是 $\sum_i \sum_j a_i b_j (x-1)^i \left(1-x^2\right)^j x^{n-2j} (n-i-j-1)!$

如果记 $H(u,v) = \sum_i \sum_j a_i b_j (n-i-j-1)! u^i v^j$，那么答案就是 $x^n H\left(x-1, x^{-2}-1\right)$，可以直接使用 $\mathcal{O}(n)$ 次卷积计算，总复杂度就是 $\mathcal{O}\left(n^2 \log n\right)$。