T1 排列

题目难度：中等。

题目标签：dp，状态压缩。

算法一

直接暴力枚举所有子集即可。

期望得分 20pts。

算法二

容易发现两个在位置上相邻且在数值上相邻的数，不需要枚举 $2^2=4$ 种情况，只需要枚举三种情况，即这两个数一共选了 $0/1/2$ 个。

直接暴搜即可，时间复杂度 $O(3^{\frac{n}{2}})$。

结合算法一期望得分 40pts。

算法三

考虑把特殊性质 A 拓展到一般的情况。

先考虑一个一般的 dp，设 $dp_{i,S}$ 表示考虑完了前 i 个数（这里指的是 $\{j|p_j\le i\}$），并且选了集合 S 中的数时的最小逆序对数以及方案数。容易发现 S 的状态数是可以压缩的，假设有一个连续的区间 $[l,r]$ 满足 $\forall j\in[l,r],p_j\le i$，那么与特殊性质 A 类似，不需要记录这个区间具体选了哪些数，只需要记录这个区间选了几个数，于是对于每个极长的连续段，都只记录选了几个数即可。容易证明这样的状态数是 $O(3^{\frac{n}{3}})$ 的。

具体实现可以对每个 i 预处理出 $p_j\le i$ 的连续段，然后用一个混合进制数表示状态，预处理位权之后容易做到 $O(1)$ 转移。

时间复杂度 $O(n3^{\frac{n}{3}})$。

期望得分 100pts。如果转移复杂度写大了也有 80pts。

T2 整除

题目难度：中等。

题目标签：结论，数据结构。

记 $A=\sum_{i=1}^{n}c_ix^{a_i},B=\sum_{i=0}^{m-1}x^i$。

算法一

容易证明 $x>n$ 时要么全部满足条件要么全部不满足条件。于是直接枚举 x 然后快速幂计算 A 对 B 取模判断是否为 0 即可。

期望得分 10pts。

算法二

当 a_i 较大且 m 较小时可以先做一步取模，容易证明 $x^a \equiv x^{a \mod m} (\mod B)$，这样就可以让 $a_i < m$。显然当 $a_i < m$ 时，$-nB \le A \le nB$。于是直接枚举 x 然后枚举 $i \in [-n, n]$ 判断是否 $A + iB = 0$。判断 $A + iB$ 是否等于零是简单的，令 $A + iB = \sum_{i=0}^{m-1} f_i x^i$，直接从大往小枚举 i，维护一个 num，每次 $num \leftarrow num \times x + f_i$，如果过程中 $|num| > 10^9$ 就直接退出，最后看 num 是否等于零即可。

时间复杂度 $O(n^2 m)$，但是跑不满可能能过子任务 3。

期望得分 30-40pts。

算法三

考虑用另一种方法判断算法二中的 $A + iB = 0$。找一个大质数做哈希，于是变成了 $-AB^{-1} \in [-n, n]$，B 可以快速算出，A 需要 $O(n \log m)$ 算出。

时间复杂度 $O(n^2 \log m)$。期望得分 50pts。

算法四

前面的算法二和算法三都很难低于 $O(n^2)$。(不过很有乱搞的潜质，如果有人乱搞过了也是正常的)

我们发现 B 的性质很不好，因为 B 的倍数没有什么规律。考虑先做一步转换，另 $A' = A(x-1), B' = B(x-1)$，显然 A 能被 B 整除等价于 A' 能被 B' 整除。这个时候 $B' = x^m - 1$，有很好的性质。

首先有一个结论，若一个数 $C = \sum_{i=0}^{m-1} f_i x^i (|f_i| < x)$，则 C 能被 B' 整除当且仅当 $\forall i, f_i = 0$ 或 $\forall i, f_i = x - 1$ 或 $\forall i, f_i = -(x-1)$。证明是显然的，考虑 $-B' \le C \le B'$，如果 $C = 0$，那么显然所有 f_i 都得是零，如果 $C = B'$，那么由于这时 C 是所有满足 $|f_i| < x$ 的数中最大的，所以必然所有 $f_i = x - 1$，如果 $C = -B'$ 也同理。

有这个结论之后就好做了，考虑把 A' 变成上文中 C 的形式，然后就可以快速判断了。首先特判掉 $x = 1$，然后枚举 $x > 1$，我们可以通过有限次进位使得 $|f_i| < x$，具体的，每次找到一个 i 使得 $|f_i| \ge x$ 然后 $f_i \leftarrow f_i \pm x, f_{(i+1) \mod m} \leftarrow f_{(i+1) \mod m} \pm 1$，由于每次进位会让 $\sum |f_i|$ 减少 $x - 1$，所以最多只会进位 $O(\frac{n}{x-1})$ 次。具体实现可以用一些数据结构快速找到绝对值最大的 f_i 进行进位，每个 x 计算完之后都进行撤销。

一共只会进行调和级数次进位，一次进位可以 $O(\log n)$。时间复杂度 $O(n \log^2 n)$。期望得分 100pts。数据结构用的不好或者常数过大也能通过子任务 4 或子任务 5。

T3 消逝的传承

难度：困难。

算法标签：计数/ad-hoc/容斥/NTT

算法一

直接 $O(2^{\frac{n(n+1)}{2}})$ 枚举每个 T 是啥然后暴力 chk，可以通过子任务一。期望得分 10pts。

算法二

考虑 dp，设 $f_{i,s}$ 表示考虑了 $T_0, T_1, \ldots, T_i$，其中 $T_i = s$ 的方案数，转移 $O(n)$，复杂度 $O(2^n n)$，写的再丑也能过子任务二，期望得分 20pts。

算法三

注意到我们需要考虑每次加入一个数字的时候不能重复计算，我们要求每一个 `0` 不能加在原来的 `0` 的前面，`1` 不能加在 `1` 前面，那么每个位置都恰好有一个数字不能加，最后一个位置两个数字都能加，因此从 T_{i-1} 到 T_i 的方案数是 $(i+2)$。

对于全是 `?` 的情况，相当于对 T_n 没有限制，答案即为 $(n+1)!$。可以通过特殊性质 A，结合前面期望得分 25pts。

算法四

注意到从前往后加数不容易处理最后对 T_n 的限制，因此我们从后往前删数，每次要求删除某一个连续段的最后一个。但这样还是难以处理限制。

考虑一个天才想法，考虑所有 `0` 的段从后往前删，所有 `1` 的段从前往后删，这样我们删除的位置永远都是 `01` 交界处（如果我们再在左侧虚拟一个删不掉的 `0`，右侧虚拟一个删不掉的 `1`）。这样的好处是什么：我们每次只需要找到一个 `01` 交界处，然后把 `01` 吃掉换成一个 `0/1`，这可以看作一个 `?`。这样如果全都是问号我们就相当于每次吃掉两个相邻的问号，此时会要求左侧为 `0` 右侧为 `1`，相当于确定了 `?` 的取值，然后变成一个全新的 `?`。如果是两侧相当于直接删掉一个 `?`，这样从 T_i 删到 T_{i-1} 的方案数是 $i+1$，总方案数就是 $(n+1)!$，得到了和算法三一样的结论。期望得分 25pts。

算法五

但是算法四有非常大的扩展性。我们考虑问每个数在什么时候被删除，每次删除的左侧是 `0/?`，右侧是 `1/?`。但这个实际上限制对问号是没有的，只有对 `01` 有。但是我们 `01` 参与过一次删除之后，它对左右两侧的限制就会消失，因为它会变成一个 `?`。这启发我们考察每个空隙的删除时间，话句话说，我们考虑每次删除时选择一个 `01`，删掉之后换成问号，现在改成每次选择一个 `01`，删掉之后换成两个问号，并且对中间这个空隙标记删除。

这样我们对着 `n+1` 个空隙选择删除顺序即对应了一组方案，那么每个数字 `0` 的限制即为要求其左侧的空隙比右侧的晚删，`1` 的相反。这样我们把原题的限制变成了给定一个 长度为 n 的串，每个位置可以是 `</>/_`，问有多少个长为 $n+1$ 的排列满足这些限制。这个实际上就是 loj 不等关系。接下来的算法都是在解决这个问题。

对于特殊性质 B，我们把每个 `</>` 的左右两边的位置标记为关键点，剩下的排列可以乱选，对这部分的限制状压解决，假设有 k 个 `</>`，复杂度为 $O(2^{2k})$。期望得分 50pts。

算法六

考虑 dp 计算方案数，设 $f_{i,j}$ 表示填了前 i 个数，其中第 i 个数在前 i 个的排名是 j，转移枚举第 $i+1$ 个数填了啥，直接暴力做是 $O(n^3)$。期望得分 60pts。

算法七

对上述算法进行前缀和优化，复杂度 $O(n^2)$，期望得分 70pts。

算法八

我们通过 `?` 会把序列分成若干段，每一段相对独立，我们对于每一段统计方案数，最后使用组合数把这些段组装起来即可。

同时出现 `<` 和 `>` 会让我们很难办，因此我们考虑把 `>` 容斥掉，限制变成一堆 `<`，方案数就是 $(n+1)!\prod \frac{1}{i!}$。那么对于每个 `>` 相当于有两种情况：可以变成 `<` 有 `-1` 的贡献，或者把这里断开。

那么我们直接枚举每一段，然后转移。设 f_i 表示我们考虑了前 i 个限制的贡献，那么我们要求 $s_i =$ `>` 转移考虑下一段的长度，那么有：

$$f_i = \sum_{s_j=>} f_j \frac{1}{(i-j)!} \times (-1)^{c(j+1,i)}$$

其中 $c(l,r)$ 表示区间 $[l,r]$ 内的 `>` 个数。

直接暴力做也是 $O(n^2)$，期望得分 70pts。

但是这玩意是：

$$(-1)^{c(1,i)} f_i = \sum_{s_j=>} (-1)^{c(1,j)} f_j \times \frac{1}{(i-j)!}$$

是个半在线卷积的形式，使用分治 NTT 可以做到 $O(n \log^2 n)$。期望得分 100pts。