2022CCPC 网络赛题解

2022 年 9 月 11 日

1 Doubt VS Lie

模拟质疑牌游戏。每次记录当前所放牌真假，当有玩家质疑时，根据真假，将当前桌上牌，收给败者。

2 Degree

该题目是在一个无向图中找出一个导出子图，使得子图中的点的度数和原图中模 3 同余。

首先可以想到如果原图不连通则任选一个连通块都可以作为答案。其次很显然如果有点的度数为 3 的倍数，则单独选这一个点也可以构成答案。

由此，只剩下图中的点度数模 3 为 1 和 2 的情况。那么显然，如果度数模 3 为 2 的点可以构成环且这个环并不是整张图，这个环就可以作为答案。倘若有不止 1 个度数模 3 为 1 的点，任选其中两个点相连，只要这条连线不是整张图，也可以作为答案。

这之后就只剩下度数模 3 余 1 的点数为一个，剩余的点皆是度数模 3 余 2 的点的情况。在此情况下，整张图的结构必然是若干个由度数模 3 余 2 的点构成的环相交在度数模 3 余 1 的点上。在这情况下，选出这一个交点，在任选两个过这一个交点的环就可以作为答案。

3 Guess

将两个人的数字分别表示为 $pt, 2pt, t$ 是奇数，p 是 2 的整数次幂。可以注意到无论 t 如何取值，对游戏过程是没有影响的，不妨再假设 $t = 1$。

若 Alice 看到 Bob 的数字是 1，可以接着报出自己的数字是 2。

若 Bob 看到 Alice 的数字是 1，说明自己的数字是 2，这时 Alice 不知道自己是 1 还是 4，所以只能 “I don’t know”，Bob 就可以报出自己的数字了。

若两人的数字分布是 $2, 4$，看到对方是 4 的人，因为不确定自己是 2 还是 8，所以只能说 “I don’t know”，这就帮助对方确定了自己是 4，因为他自己可能是 1 或 4，而 1 的可能被排除了。

若两人的数字分布是 $4, 8$，看到对方是 4 的人，因为不确定自己是 2 还是 8，所以只能说 “I don’t know”，而对方看到的是 8，也不知道自己的数字，也只能说 “I don’t know”，这样拿到 8 的

人就排除了自己是 2 的可能了。

若两人的数字分布是 $8, 16$，随着前面几轮结束，而拿到 8 的人说出 “I don’t know”，拿到 16 的人可以排除自己是 4 的可能，从而报出自己的数字。

依此类推。

因此：一定是数字更大的人先知道自己的数字大小，通过分析数字较小的情况即可找到与轮数相应的规律。

单次询问的时间复杂度 $O(\log a)$ 或 $O(1)$。

4 Pointer Sequence

1. 操作一要求区间复制，可以考虑用可持久化 Treap 来实现。可持久化 Treap 的空间复杂度较高，可考虑开一个内存池，把不用的结点回收掉，这样空间复杂度为 $O(n)$。

2. 操作二要求区间赋值等差数列，我们可以把这个操作懒处理。具体地，可以在 treap 上加一个标记表示分配的指针编号，然后把这个区间以及对应 k、b 等参数存到一个数组中。在操作五查询的时候，只需要在 treap 中找到分配的指针编号，然后在这个数组里二分即可。

3. 操作三可以独立于其他操作单独处理，我们把操作三离线，把 L、R、v 一起离散化，倒着处理每一个操作，用可持久化线段树或其他数据结构记录每一个值下一次修改在什么时刻，所有的修改构成了一个树形结构。这样在正序处理的过程中，对于一个查询操作，我们先找到这个查询的不经过操作三的原始值，然后基于上述数据结构找到下一个修改的时刻，再在树上一直走到小于当前时刻的最近一次修改。整个沿着树往上找的过程可以用倍增，或者并查集等数据结构优化。

4. 操作四进行单点赋值，开一个哈希表记录下每个指针的当前值以及最后一次赋值的时刻即可。

5. 操作五是唯一的查询操作，对于这个查询我们先查找操作四的哈希表，如果哈希表里没有，我们去操作二的数组里二分得到这个指针的初始值。有了初始值在考虑操作三对值的整体修改，得到最终的值。

整个算法时间复杂度 $O(m \log n + m \log m)$，空间复杂度 $O(n + m \log m)$。

5 Substring Match

首先是对于此类正则匹配并不是匹配越长越好，因为可能导致后面不能被匹配，然后就是题目数据范围里的保证 t 中的大写字母不会超过 200 个。

记 $dp[i][j]$ 为模式串 t 匹配到 i，匹配串 s 匹配到 j，最靠前的起始位置是多少。

考虑两种情况进行转移：

1. i 位置匹配多个对应小写字母：$dp[i][j] = \min(dp[i][j], dp[i][j-1])$ ，其中 $s[j] = \text{lowercase}(t[pos[i]])$，$pos[i]$ 为第 i 个大写字母位置。

2. 从上一个大写字母匹配中间一段小写字母转移过来：$dp[i][j] = \min(dp[i][j], dp[i-1][j-len])$，其中 len 为中间小写字母段长度，$s(j-len+1, j) = t(pos[i-1]+1, pos[i]-1)$。

   然后从 dp 数组中得到最后答案。复杂度 $O(200|s| + |t|)$.

6 Xor Circle

1. 假设有个圆就是取到最优解的圆。那么可以通过调整法，使得这个圆的边界落在某个点上。
2. 枚举这个点 i，钦定这个点必须落在圆的边界上。
3. 显然，圆的半径越大，那么异或和的最大值就越大。因此，可以二分半径 r，然后判断这个 r 是否合法。
4. 圆心 P 落在以点 i 为圆心，r 为半径的圆周上。对于其他点 j，要使得其落在圆 P 内，那么 P 的合法位置就是这个圆周上的一段圆弧。这个圆弧可以使用三角函数求出，表示为一段区间。设这段区间的权值为点 j 的权值。
5. 那么变成如下问题：给定圆弧上的若干区间以及其权值，问你是否存在一个位置，使得在包含该位置的区间中选出一些的异或和 $\geq k$。
6. 离散化后，使用线性基维护。使用线段树分治是两个 log 的，可以记录每个数的删除时间，然后插入线性基的时候优先选删除时间久的值去插入，就可以减少一个 log。
7. 回顾一下，我们需要：枚举每个点 i ，二分每个点半径 r，离散化后使用线性基维护。总的时间复杂度 $O(n) \times O(\log V) \times O(n(\log n + \log a))$。其中 $\log V$ 指二分次数。应该不足以通过此题。
8. 设 $F(i)$ 表示点 i 的最小半径。算出一个 $F(i)$ 的复杂度是 $O(n(\log n + \log a) \log V)$ 的，但判断某个 $F(i)$ 是否小于某个数的只需要一次二分的操作，也就是 $O(n(\log n + \log a))$ 的。
9. 随机打乱这些点，设当前的答案为 Ans，按照随机后的顺序枚举每个点，判断该点答案是否小于 Ans：如果小于，则使用二分算出这个半径；否则直接跳过这个点。最终需要二分算出 $F(i)$ 精确答案的次数期望是 $O(\log n)$ 的。每个点 i 都需要一次判断是否有 $F(i) < Ans$。

   因此总的时间复杂度为 $O(n(\log n + \log a) \log V \log n + n^2(\log n + \log a))$，足以通过此题。

7 Name the Puppy

我们考虑最优答案串，对于它的最长 anti-border 可以分为两种情况讨论。

如果最长 anti-border 的长度小于答案串的一半，这种情况比较好解决，我们可以把后一半长度翻转过来和前缀进行匹配，那么问题变成了所有串拼接的串与所有反串拼接的串，最长可以匹配多长。

对所有串建立一棵 Trie 树，前缀可以由该 Trie 树生成，对所有串的反串建立一棵 Trie 树，那么后缀也可以由该 Trie 树生成。因此，我们可以设 $dp(x,y)$ 表示前缀 Trie 树走到 x 节点，后缀 Trie 树走到 y 节点时最长的匹配长度，根节点 0 表示空串也可以表示某个串的匹配结束，那么答案即为 $dp(0,0)$，用记忆化搜索，枚举下一个字符的转移即可，如果重复走到某个节点则答案为 INF。

如果最长的 anti-border 的长度大于等于答案串的一半，我们会发现如果答案串的中间是跨越了某个字符串 s_i，这似乎难以被 Trie 树 DP 的办法处理。但事实上，我们依然可以想办法归到上一种情况的处理。首先这种情况其实并不存在，因为该答案串必然是一个回文串，那么我们不断重复这个串就可以得到 INF 的答案。而在 Trie 树 DP 的处理办法里，我们会发现，只需要在原本答案串中间的地方，即在前缀的最后面以及后缀的最前面不断重复 s_i，就可以让匹配不断延长，从而判断输出 INF。

最后的时间复杂度为 $O((\sum n)^2 \times \sum)$，但这个做法实际上有 $\frac{1}{4}$ 的常数，因此也可以通过本题。但要优化到 $O((\sum n)^2)$ 也并不难，只需要用链表优化跳过 Trie 树中无效的儿子即可。

8 Mutiple Set

令 $l = \lceil \frac{L}{x} \rceil, r = \lfloor \frac{R}{x} \rfloor$，则题目所求集合即为 $\{ix | l \le i \le r\}$。不难推导出 $K = 2^{r-l-1}x(l+r)(r-l+1)$。考虑枚举 K 的所有因子进行验证，若直接暴力枚举，复杂度为 $O(T\sqrt{K})$，不能通过。于是对 K 进行质因数分解，而后通过枚举指数找出 K 的所有因子进行验证。质因数分解可以采用 Pollard-rho 算法，也可以预先筛出 10^7 以内的质数，而后用这些质数对 K 做质因数分解。

9 Transport Plan

记在 i 城市与 j 城市位置建造传送门为 (i,j) ，考虑一个运输计划 a_i, b_i, v_i 对 $ans(i,j)$ 的影响：

$$ans(i,j) += v_i \times min(|a_i - b_i|, dis((a_i,b_i),(i,j)))$$

其中 dis 为平面上两个点的曼哈顿距离。将上述式子重写一下：

$$ans(i,j) += v_i|a_i - b_i| - max(0, |a_i - b_i| - dis((a_i,b_i),(i,j)))$$

前面这一项都加到最后的答案里，后面这一项在纸上写一写，发现可以先横着差分两次，再分别竖着差分和斜着差分就可以做到 $O(1)$ 修改，最后再前缀和回来就行，求一个全局最小值。可能有一些实现细节。复杂度 $O(n^2 + m)$。

10 Count Permutation

首先，容易知道 w 的最小值是 2，我们设这两个差值分别是 $p, -q (p, q > 0)$。

下证一个重要的引理：$p + q = n$ 并且 $\gcd(p, n) = 1$。

证明：

易知 $\gcd(p,q)=1,(p+q)|n$. 以下在此基础上证明 $p+q=n$.

假设 p 出现 x 次，那么 $-q$ 出现 $n-x$ 次，由总和为 0 可以解出 $x=\frac{nq}{p+q}$，$n-x=\frac{np}{p+q}$

假设命题不成立，观察到如果一个长度为 $p+q$ 的连续段出现了 p 次 $-q$ 和 q 次 p，那么这个长度为 $p+q$ 的区间和为 0，与序列是一个排列矛盾。另外，每个长度为 $p+q$ 的区间和是 $p+q$ 的整数倍。我们从第一个长度为 $p+q$ 的区间开始，每次将区间平移一格，那么区间和的变化只能是 $p+q,0$ 或者 $-(p+q)$ ，那么由介值原理，所有长度为 $p+q$ 的区间和都小于（或大于）0，由于 $(p+q)|n$，这样这些数总和不是 0，矛盾。而 $\gcd(p,n)=1$ 是显然的，引理证毕。

这样一来，整个排列构成模 n 的等差数列。接下来只需要求解满足 $(m-1)\times d=(t-s)\pmod n$ 且 $\gcd(d,n)=1,(1\le d<n)$ 的 d 的个数。可以用容斥、莫比乌斯反演等方法解决。

11 Roulette

注意到：5000 次都输的概率是 $0.99^{5000}\approx 1.5\times 10^{-22}$，也就是说至少有一次取胜。

而总钱数很大，因此我们可以：第 1 次押 y 元，第 2 次押 $2y$ 元，第 3 次押 $4y$ 元，以此类推，保证每次如果赢了就能把之前所有输的总和加上 y 元赢回来即可。

对于 C/C++ 选手，使用每次乘 10 可以规避写高精度。

——“我可以输很多次，但只要赢一次就够了。”

12 Tree Division

问题转化一下，求删除尽量多的边，使得所有连通块大小大于等于 k。

现在对于一个 k，我们有一个简单的贪心算法，即从叶子开始看，每当子树大小大于等于 k 则断掉其与父亲的边，分出一个连通块，然后一直做下去。

可以发现这样找到的所有的当前子树大于等于 k 然后断掉其与父亲连边的点总共不超过 n/k 个。

如果能快速的找到深度最低的子树大小大于等于 k 的点，并能在删掉之后更新其祖先的子树大小，那么就能解决该问题，但会发现如果直接这样去做，很难有一个很好的办法维护上述信息。

我们考虑用 $dsu-on-tree$ 和线段树来解决该问题，我们用线段树维护在点 x 时，当 k 等于 1 到 $size[x]$ 时子树中剩下的点的个数，每次先继承重儿子的信息，然后对于所有轻儿子的信息，留下的点的个数加起来之后，直接更新到线段树中，然后对于大于最大轻儿子大小的所有 k，其那些轻儿子所留下的点的个数都是其对应的原始子树大小，所以轻儿子部分的留下的点的个数和是定值，可以通过线段树的区间加操作来进行维护，然后线段树中需要记录的是其中 $k-left[k]$ 最小的，即余下空间的最小的一个 k，如果小于等于 0，则对其进行操作，对 $ans[k]$ 进行加一，并将其在线段树中维护的子树大小清零，然后继续做下去，直到线段树中没有留下的点数超过其对应限制的 k 了，则完成 x 点的信息维护。

因为前面进行了一个问题转化，再将得到的答案转化一下，得到原问题的答案。

复杂度 $O(n\log^2 n)$。

13 Count Permutation 2

签到题。

显然只有 n 在排列中的位置是有用的。

一次操作中至少要有一个 n 才是有意义的。

一次操作中至多可以使得 $k-1$ 个原来不是 n 的位置变成 n。

初始情况一共有 $n-1$ 个位置不是 n。那么答案的下界为：$\left\lceil\frac{n-1}{k-1}\right\rceil$。

通过调整，可以发现一定能够取到下界。

一种证明：将这些位置分段，分成 $[1,k],[k,2k-1],[2k-1,3k-2],\cdots$。每段长度为 k，相邻两段之间有一个重合位置。找到 n 所在段，然后对这个段进行一次操作，接着往左右两边扩展即可。

于是连排列都不需要输入，直接输出 $\left\lceil\frac{n-1}{k-1}\right\rceil$ 即可。

时间复杂度 $O(1)$。