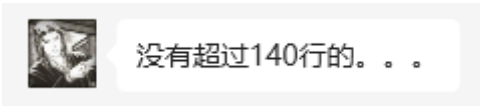
数据结构选讲（一）

Inaba Meguru

2023.10.18



A. Copium Permutation（10）

牛逼题。

下面称作题目中的「Copium 段」为「好段」。

考虑如果 $k=n$，那么这个题就是 CF536F。我们觉得 CF536F 应该是这道题的一个部分。

我们先考虑对一个固定的 k 怎么用多项式的时间算出答案。我们把序列分成两段，一段是前面 k 个元素，另一段是后面 $n-k$ 个元素。我们称后面 $n-k$ 个元素为「任意元」。这两段内部的答案都是好求的：

- 前面的可以通过 CF536F 的方法求。（这道题可以以 $\mathcal{O}(n\log n)$ 的方法求每一个前缀）。
- 后面的部分我们大胆猜测最优情况肯定是每个极长的任意元连续段都一起摆放。（如任意元为 $1,2,3,6,8$，肯定会把 $[1,2,3],[6],[8]$ 一起摆）通过 `set` 可以以 $\mathcal{O}(n\log n)$ 求解每一个 k。

横跨两边的部分也可以求。我们只要最大化左端点在 $[1,k]$ 右端点在 $[k,n]$ 之间好段的最大值。

由于左端点到 k 的部分是固定的，我们考虑从左端点入手。

首先怎么样的左端点可能有一个匹配的右端点呢？显然是对 $[1,k]$ 的元素离散化后 $a[l,k]$ 是好的的所有 l，换句话说，假设 $mx=\max a[l,k], mn=\min a[l,k]$，那么要求 $\forall i<l$，都有 $a_i\notin[mn,mx]$。我们称这样的 l 为漂亮的。

考虑每个漂亮的位置，考虑怎么增加贡献。首先我们如果想要 $a[l,*]$ 增加贡献，首先必须要填入 $[mn,mx]$ 当中所有任意元。在填入所有任意元之后如果再填入 $mn-1$ 或者 $mx+1$ 又会增加贡献。

由于 mn,mx 的情况是对称的，我们只需要考虑 mx。首先如果填入 $[mx+1,ed]$（ed 是最大的使得 $[mx,ed]$ 只有任意元的值）使得一段区间的漂亮的位置都能产生贡献，我们先研究这样的区间是怎么样的：

- （A）这些漂亮位置的 mx 初始时就相等。这是因为如果初始时 mx 不等，假设这些数当中最大 m_1，最小 m_2。因为所有的 mx 都不是任意元，所以最 m_2 的左端点永远得不到 m_1 作为连续段（因为不是任意元，内部也不含 m_1）。
- （B）假设选定漂亮位置为 $p_1,p_2,\ldots,p_m$，那么要求 $a[p_i,p_{i+1}-1]$ 是好段。证明大概是放一个自由元段会使 mn 大于其的元素（除了首个）都不能继续向 mx 扩展。自己理解一下。

我们对一个 mx 的贡献讨论完毕，通过枚举 mx（和 mn）统计贡献即可做到 $\mathcal{O}(n)$ 的复杂度求解一个 k。（见附件当中 `a/bf.cpp` 或者 bf-submission）。

我们考虑怎么对每个 k 求解。首先考虑每一个漂亮的位置的变化，一个 a_k 的加入会对一个后缀的 mn,mx 进行更改，那么被剔除漂亮的位置的应当是一段原先漂亮位置的后缀，最后再加入 k。这特别像一个栈，每次删掉一坨然后加入 k。

上面的考虑维护（B），我们只需要知道这个漂亮位置和前面漂亮位置之间最大最小值即可，每次删栈要把栈顶和次栈顶合并。同时像暴力一样维护当前 mx 等价类最长段和历史最长即可。每次仅当 mx 和下一个元素发生变化或者栈顶才统计答案。时间线性。

参考代码见下发 `a/a.cpp` 或者 submission。

B. Diverse Segments (6)

简单题。

我们相当于删除一个区间使得每个线段剩余元素两两不同。考虑一个线段怎么贡献，假设一个线段内出现颜色 c 的位置为 $p_1, p_2, \ldots, p_m$（$m \geq 2$），那么我们想要 $[l, r]$ 覆盖 $[p_1, p_{m-1}]$ 或 $[p_2, p_m]$。经典的把完全重叠的线段删掉，剩下线段左端点随右端点递增。爱怎么维护怎么维护。

参考代码见下发 `b/b.cpp` 或 submission。

C. Kazaee（6.5）

我最爱的哈希/se

对每个元素随机一个权值，那么满足条件就必须这个区间的权值和为 k 的倍数。我们多随几遍就可以了。

复杂度 $\mathcal{O}(Tn\log n)$，正确率为 2^{-T}。其中 T 是随的次数。

参考代码见下发 `c/c.cpp` 或 submission。

D. Sharti (8)

结论题。

首先这种棋子翻转问题有个通常的套路，就是每个棋子的贡献应当是独立的。更加具体的，可以给每个位设一个 sg 值，使得后手胜利当且仅当所有棋子位置的 sg 值异或为 0。感性理解一下，我们翻动一个棋子会带动其他棋子翻转，那么如果这些棋子原来就有就会 sg 异或抵消掉，相当于没有。

首先不考虑 k 的限制，根据 sg 函数的定义，容易得到：

$$sg_{x,y} = \operatorname{mex}_{0\le k\le \min(x,y)} \operatorname{xor}_{i\in[x-k,x],j\in[y-k,y],(i,j)\neq(x,y)} sg_{i,j} \tag{2}$$

打个表容易发现 $sg_{x,y} = \min(\operatorname{lowbit}(x), \operatorname{lowbit}(y))$。然后加上 k 的限制不难发现 $sg_{x,y} = \min(\operatorname{lowbit}(x), \operatorname{lowbit}(y), \operatorname{highbit}(k))$。

然后怎么求题目中覆盖的 sg 的异或呢？显然考虑扫描线，每次求得覆盖区域当中 2^i 倍数的个数。即可。

参考代码见下发 `d/d.cpp` 或 submission。

本来附件里有一个 `bf.cpp` 给大家玩玩的，不见了。。。

E. A Stroll Around the Matrix (7)

这种问题首先要考虑一个容易被 ds 维护的东西或者解法。一般这个解法应当是简单的，重点应当在怎么上 ds。

首先考虑不修改的情况。手膜不难发现每次走权值比较小边的贪心就是对的。但是这玩意 ds 显然不太好维护，因为每一次加入的是一个等差数列。转到差分就是差分的后缀加，我们可以考虑维护差分。

题目说了两个序列的差分都是单调增的，上面走路径相当于每次选一个差分小的一边走过去，这个差分会给后面的所有步增加。也就是我们考虑给 a, b 的差分数组 da, db 进行二路归并得到 d，最后差分造成的贡献就是 $\sum (n+m-1-i) \times d_i$。

观察 n 非常小，应当考虑 $\mathcal{O}(nm \log)$ 的做法。我们把上面的式子拆开，分成 $\sum d$ 和 $\sum i \times d_i$。$\sum d$ 是好求的，$\sum i \times d_i$ 有一定难度，因为我们需要对 da, db 进行二路归并。考虑到 a 的长度比较小，我们可以用 ds 维护 db，同时暴力插入 da 查询贡献。

一个基本的想法是用 splay 解决。每次插入 da，查询完之后删除 da。遗憾的是虽然复杂度正确，但是 splay 常数太大，不太能做。

我们考虑到我们的 db 已经有序，我们只需要每次查询 da 在 db 当中的排名即可。用线段树维护常数小很多。

见下发 `e/e.cpp` 或 submission。