

T1 路径

定义 u 能到达 v 为存在一条从 u 开始、以 v 结束的路径。

这里定义两个点 u, v 弱连通为 u 能到达 v 或者 v 能到达 u 。

首先发现一个比较显然的性质：如果存在两个关键点不弱连通，则一定不存在满足条件的路径。此时输出 `No` 即可。

然后发现如果满足任意两个关键点弱连通，则一定有解。将所有关键点拎出来，如果 u 能到达 v ，则连一条 u 到 v 的边。注意到此时任意两个关键点都有边，即这是一个竞赛图。又因为原图是有向无环图，所以这个竞赛图也是有向无环图。

竞赛图一定存在一条经过所有点的简单路径，我们找到这样的一条路径，然后将这条路径上相邻的点在原图随便找一条路径，这些路径连起来一定就是合法的了。

关于竞赛图的性质，有兴趣可以看看[这位大佬的博客](#)。

有了这个性质，我们只需要判断关键点是否两两弱连通即可。

首先可以使用 bitset 做到 $O(\frac{n^2}{w})$ 判断，但是对于这道题很明显过不了。当然也可以 $O(nk)$ 暴力判，显然也过不了。

考虑拓扑排序，注意到如果拓扑排序的过程中如果有两个关键点同时在队列中，则一定不合法。但是只是拓扑排序显然还是不行的，会有不合法情况被误判成合法。

考虑稍微修改一下拓扑排序，从队列中取点时优先取出非关键点，然后如果有两个关键点同时在队列中就不存在合法路径，否则就存在。

发现这样做就是对的了。为什么呢？

考虑证明。令 u 为一个关键点， S 为所有能到 u 的点组成的点集去掉 u ， T 为所有 u 能到达的点组成的点集去掉 u 。当 u 入队时，所有 S 中的点都已出队，而 T 中的点都只有到 u 出队后才能入队。而在 u 入队到出队这段时间，假如有关键点与 u 不弱连通，它就一定会入队。因此如果有两个关键点不弱连通，一定会被这个特殊的拓扑排序判到。

因此跑一边这个拓扑排序就好了，判断也是简单的。

时间复杂度 $O(n + m)$ 。

当然存在别的做法，大家可以自行思考。

T2 异或

将序列转换为差分序列（设为 d ，其中 $d_i = a_i \oplus a_{i-1}$ ），就可以将区间修改变成单点修改（后缀区间修改）或双点修改（非后缀区间修改）。同样要求全部变为 0。此后所有文字均是在这个差分序列上操作。

我们暂时不考虑 w 的取值，只考虑有无合法的修改方案使得存在一组合法的 w 。

我们可以将 N 个数抽象为 N 个点，将修改操作抽象为无向边（或自环），这样一组合法的方案由若干个连通块组成。

考虑每个连通块（设大小为 x ）的最小边数，发现下界为 $x - 1$ （一棵树），上界为 x （一棵树加上一个自环必定有解）。考虑如何取到这个下界。

我们发现，可以取到这个下界当且仅当连通块内的数异或和为 0。

必要性：显然每次操作都是双点修改，那么整个连通块内的数异或和不会变化，而最终要变为 0，那么开始时必须是 0。

充分性：考虑一条串起连通块内的所有数的链。每次将链头通过异或自己的方式删掉，下一个数受到影响后成为新的链头。经历 $x - 1$ 次操作后，最后一个数肯定是 0，无需再操作。

也就是说，我们需要将差分序列划分为若干子序列，使每个数恰好属于一个子序列，每个子序列的权值即为 $x - 1$ （子序列异或和为 0 时）或 x （其它情况），求出一种划分方式使得权值和最小。

发现答案即为 N 减去异或和为 0 的子序列数，那么我们就需要最大化异或和为 0 的子序列数。先状态压缩再通过枚举子集进行动态规划即可。

最好预处理出每个子序列的异或和。

时间复杂度 $\Theta(3^n)$ 。

T3 距离

首先考虑子任务 2，此时相当于从点对变成了单个点。问题变成往点集插入一个点，或者询问距离某个点最近的点的距离。

考虑点分树。在每个重心维护其点分树子树内其最近的点的距离，查询时直接查就好了。因为保证边权为正，且求的是距离之和最小，所以直接做就是对了。复杂度 $O(n \log n)$ 。

但现在是对点，怎么办？

其实很简单，再套一层点分治就好了。将所有操作离线下来，按照 a_i 和 x_i 点分治，在这一层点分治时处理所有 a_i 或 x_i 在其点分治子树中的操作。令当前分治中心为 u ，则按时间顺序在另一颗点分树上插入 b_i ，加上 $\text{dis}(u, a_i)$ ，查询也使用普通点分树的操作直接查询就好了。

然后就做完了。点分治套点分树，时间复杂度 $O(n \log^2 n)$ 。实现起来很好写。

顺带一提，该做法大概是可以支持强制在线的，只不过空间复杂度可能要到与时间复杂度同阶，而且会比较难写。

T4 花之舞

首先我们考虑全局询问怎么做。

显然，最近点对的两个点必须删一个，所以我们要先扫描线求出最近点对，然后分别求出删掉一个点后的最近点对。一共跑 3 遍最近点对，总时间复杂度就是 $\Theta(qn \log n)$ 。

区间也是同样的做法，但是求如此多次最近点对并不现实。考虑类似 [P9062 Ynoi2002 Adaptive Hsearch&Lsearch](#) 的方式，求出支配对。考虑将平面直角坐标系划分成大小为 $2^k \times 2^k$ 的矩阵，然后对于每个矩阵，找到相邻矩阵（相邻指共边或共顶点，也即八连通）和自己的矩阵，对这九个矩阵分别进行操作。每次操作就是将这两个矩阵（自己和操作对象）的点按编号排一下，可以证明支配点对的编

号排名差不超过 6，考虑到后面要删一个点，就取到 7 或 8 为宜。记得将支配点对去重。

然后我们考虑询问怎么做。考虑依照询问右边界升序扫描线，将支配点对插入线段树里。

线段树每个节点只考虑：左端点在节点管理区间里 且 右端点不超过当前扫描到的边界 的支配点对。维护几个值，分别是：

- 这些支配点对里，最近点对是哪对，距离是多少；
- **分别**删掉上一条中最近点对中的一个点后的最近点对距离。

合并区间的时候，设两边最近点对分别是 (u_1, v_1) 和 (u_2, v_2) ，其中 $u_1 < v_1, u_2 < v_2$ 。如果两边最近点对有共同端点，就可以将 删掉这个共同端点后的最近点对距离 更新为两边最小值；如果两边最近点对没有共同端点，那么一边的最近点对**可能**更新 另一边的删掉某点后的最近点对距离。

询问时考虑删掉最近点对的哪个点，取最优即可。

时间复杂度： $\Theta(n \log n \log w + q \log n)$ ，其中 w 为坐标最大值（本题中是 1×10^8 ）。可能会带点常数。

另外存在一个整体二分的做法，时间复杂度大概 3 个 $\log$ 且写起来更复杂。