省选 Day1 题解

2026 年 3 月 15 日

给定一棵 n 个点的以 1 为根的树，定义一种**随机剖分**算法，流程如下：

- 对树进行自底向上的处理。每个点 u 要选出一个子节点作为**重子节点**，之间相连的边为**重边**，其余边为**轻边**。所有重边构成了若干条路径，称为**重链**。
- 对于点 u，设其有 k 个子节点 $v_1, \ldots, v_k$，对应与之相连的重链长度为 $l_1, \ldots, l_k$。则点 v_i 成为重子节点的概率为 $\frac{l_i}{\sum_{j=1}^{k} l_j}$。

设 c_u 表示在上述随机剖分的算法下点 u 到点 1 的最短简单路径上轻边的数量，请求出 $\sum_{j=1}^{n} c_j$ 的**期望**，对 998244353 取模。

数据范围：$n \leq 5000$。

由于期望的线性性，我们可以将问题转化为求每条边作为轻边的概率。设点 i 到其父节点 fa_i 之间的边为 e_i，边 e_i 为轻边的概率为 P_i，点 i 子树内节点个数为 siz_i，则答案为：

$$\sum_{i=2}^{n} P_i \times size_i$$

我们不妨从完全二叉树入手（虽然这个特殊性质并没有出现在数据范围中）。

设 $f_{u,i}$ 表示自底向上处理到节点 u 后，u 所在重链长度为 i 的概率。若 u 为叶子，则 $f_{u,1}=1$，$f_{u,i}=0,\forall i\neq 1$。

设 u 的两个子节点为 l,r。我们可以枚举两个子节点的重链长度 x,y，则有：

$$f_{u,x+1} \xleftarrow{+} f_{l,x}\times f_{r,y}\times\frac{x}{x+y}$$

$$f_{u,y+1} \xleftarrow{+} f_{l,x}\times f_{r,y}\times\frac{y}{x+y}$$

对应的，我们也可以统计 e_l, e_r 作为轻边的概率：

$$P_l \xleftarrow{+} f_{l,x} \times f_{r,y} \times \frac{y}{x+y}$$

$$P_r \xleftarrow{+} f_{l,x} \times f_{r,y} \times \frac{x}{x+y}$$

其实就是将系数对调了一下。

每个状态 $f_{u,i}$ 中 i 的取值范围为 $[1, md_u]$，其中 md_u 表示 u 子树的高度。合并的时候，我们直接枚举两个子节点的长度，乍一看是 $O(n^3)$ 的。实际上是 $O(n^2)$ 的，证明见正解。

我们尝试将二叉树的方法拓展到三叉树上。设点 u 的三个子节点的重链长度分别为 a, b, c，我们希望先合并 a 和 b，然后将“a, b 合并得到的整体”与 c 合并，最终得到一个重子节点。这里的合并指的就是通过类似 $\frac{x}{x+y}$ 的结构决出谁成为重子节点。

a, b 合并中 a 胜的概率为 $\frac{a}{a+b}$。我们知道 a 成为 a, b, c 当中的重子节点的概率为 $\frac{a}{a+b+c} = \frac{a}{a+b} \times \frac{a+b}{a+b+c}$，前者是 a 在 a, b 之间胜出的概率，后者是一个大小为 $(a+b)$ 的链与 c 胜出的概率。因此，“合并 a, b”相当于暂时将其视作一个大小为 $(a+b)$ 的整体。如果这个整体在后续的合并中胜出，我们再乘上 a, b 两者之间胜出的概率。

对于一般情况，设 u 有 k 个子节点 $v_1, \ldots, v_k$，对应的重链长度为 $s_1, \ldots, s_k$。我们依次对于 $i = 2, \ldots, k$，对 $(\sum_{j=1}^{i-1} s_j)$ 和 s_i 进行合并。那么 v_i 能成为重儿子的条件就是以下三点：

- 每个 v_j 的重链长度就是对应的 s_j；
- s_i 与 $\sum_{j=1}^{i-1} s_j$ 合并中胜出。
- $\sum_{j=1}^{i} s_j$ 在后续的合并中胜出。

我们很难直接计算它们发生的概率。但是，我们可以将第一个条件中 $j \leq i$ 的条件并到第二个条件中，剩下的并到第三个条件中。在第三个条件中，我们可以视 $\sum_{j=1}^{i} s_j$ 是一个定值（即不需要计算它发生的概率）。此时，两个条件成立的概率是独立的，可以分别计算。

我们先从第二个条件入手：

- $\forall j \le i$，v_j 的重链长度为 s_j，且 s_i 与 $\sum_{j=1}^{i-1} s_j$ 合并中胜出。

不妨设 $S_{i-1} = \sum_{j=1}^{i-1} s_i$。则 s_i 合并中胜出的概率为 $\frac{s_i}{S_{i-1}+s_i}$，v_i 长度为 s_i 的概率为 f_{v_i,s_i}。

由于概率的式子和单个 $s_j, j < i$ 的具体值无关，而只与其和 S_{i-1} 有关。所以，我们其实只需要得知前 $i-1$ 个子节点的重链长度总和为特定的 S 的概率 $g_{i-1,S}$。这个概率涵盖了所有不同的 $\sum_{j=1}^{i-1} s_j = S$ 的可能。

$g_{i,S}$ 的转移是容易的。初始条件为 $g_{0,0}=1$。然后有转移：

$$g_{i,S} \xleftarrow{+} g_{i-1,S-s_i} \times f_{i,s_i}, \forall S, s_i$$

于是第二个条件成立的概率就是 $g_{i,S_{i-1}} \times f_{i,s_i} \times \frac{s_i}{S_i}$。

然后考虑第三个条件：

- S_i 在后续的合并中胜出。

我们可以倒过来做。设 $h_{i,S}$ 表示前 i 个子节点的重链长度为 S，在后续的合并中胜出的概率。那么第三个条件成立的概率就是 h_{i,S_i}。

转移的初始条件即 $h_{k,S} = 1, \forall k$，这是因为后续不需要再进行任何合并了。转移：

$$h_{i-1,S} \xleftarrow{+} h_{i,S+s_i} \times f_{i,s_i} \times \frac{S}{S+s_i}, \forall S, s_i$$

当我们算出了所有的 g, h，我们就能计算出“第 i 个儿子，重链长度为 s_i，且第 i 个儿子成为重子节点”的概率：

$$f_{u,s_i+1} \xleftarrow{+} \sum_{S} g_{i,S-s_i} \times f_{i,s_i} \times \frac{s_i}{S} \times h_{i,S}$$

用 f_{i,s_i} 减去上面的式子就是“第 i 个儿子，重链长度为 s_i，且第 i 个儿子不是重子节点”的概率。将不是重子节点的概率加起来即得 P_u。

我们来分析一下复杂度。设 md_u 为 u 子树的最大深度，则状态 $f_{u,i}$ 中 i 的取值范围即 $[1, md_u]$。对应的，对于一个节点 u，$g_{i,S}, h_{i,S}$ 中 S 的取值范围即 $[1, \sum_{j=1}^{i} md_{v_j}]$。

那么加入 u 的第 i 个子节点的复杂度为 $O((\sum_{j=1}^{i-1} md_{v_j}) \times md_{v_i})$。处理 u 的总复杂度为：

$$\begin{aligned} T(u) &= O(\sum_{i=1}^{k}(\sum_{j=1}^{i-1} md_{v_j}) \times md_{v_i}) \\ &= O(\sum_{i \neq j} md_{v_i} \times md_{v_j}) \end{aligned}$$

不妨假设所有子节点的 md 中，md_{v_1} 是最大的，那么：

$$T(u) = O(md_{v_1} \times \sum_{j=2}^{k} md_{v_j} + \sum_{\substack{i \neq j \\ i,j>1}} md_{v_i} \times md_{v_j})$$

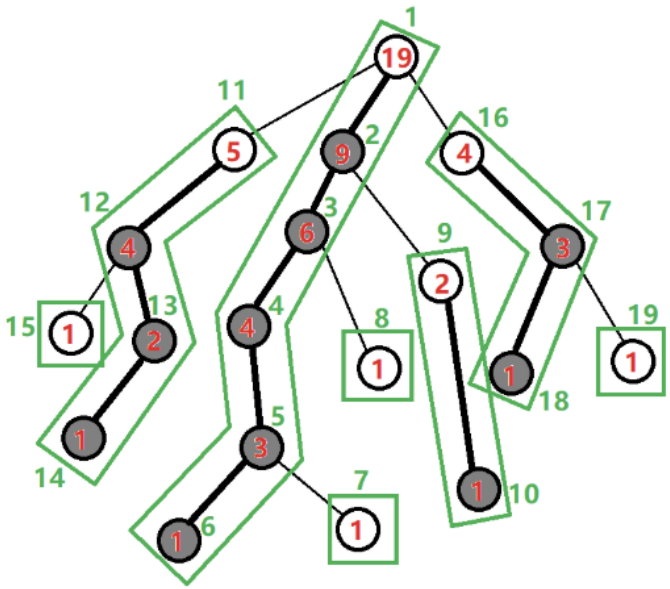
$$\leq O(n \times \sum_{j=2}^{k} md_{v_j} + (\sum_{j=2}^{k} md_{v_j})^2)$$

可以证明，$\sum_u \sum_{j=2}^{k} md_{v_j} < n$，那么总复杂度就是 $O(n^2)$ 了。

如右图，我们对树进行长链剖分，即将每个 u 与其 md 最大的子节点 v_1 之间连一条边，形成了若干条图中绿色的长链。那么，每个 $md_{v_i}, i \geq 2$ 都对应着一条以 v_i 为端点的长链的长度。

由于每个点至多被一条链包含，所以有 $\sum_u \sum_{j=2}^{k} md_{v_j} = n - md_1 < n$。

于是复杂度得以保证。



图：长链剖分。图源：oi-wiki

图中绿色的标号为顶点标号。

给定一个长度为 n 的 01 串 s 和一个整数 k。定义一个长度为 m 的 01 串 t 是**摩卡串**当且仅当：

- s 是 t 的子串；
- t 中**恰好**有 k 个子串**字典序严格小于** s。

请找出最短的摩卡串，或报告无解。

多测，$T \leq 5$，$n \leq 200$，$k \leq 3000$。

我们先研究一下什么串 t' 的字典序比 s 小。可以发现我们可以将其归为两类：

1. t' 是 s 的真前缀；
2. 存在一个 $i < n$ 满足 t', s 的最长公共为 i 且 $s_{i+1} = 1$，$t'_{i+1} = 0$。这种情况下无论往 t' 后面接什么字符，得到的新串的字典序也严格小于 s。

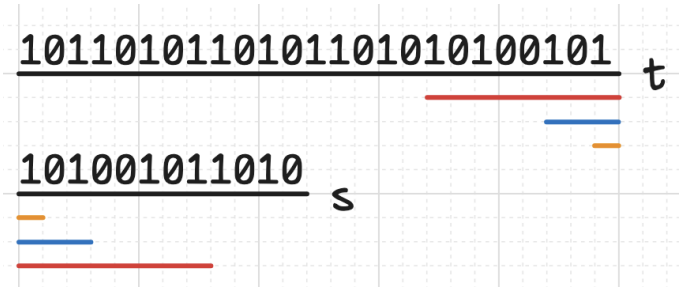
这启发我们令 t 从空串开始，每次往 t 的末尾加入一个字符，最终达到目标。

观察一下在这个过程中我们需要维护什么。首先我们要维护当前 t 的长度 $|t|$ 和当前字典序 $< s$ 的子串个数 c。我们还要维护当前 s 是否已经作为完整子串出现在了 t 当中。

对于新增的二类子串，假设其中有 b 个是当前 t 的后缀，那么无论往 t 后面加什么字符，字典序 $< s$ 的子串个数都会增加 b（并且这 b 个新的子串一一对应了原来的 b 个后缀），因此也要维护这一个量。
对于新增的一类子串，它们不仅是当前 t 的一些后缀，它们同时是 s 的一些前缀。

如右图，当前 t 有 3 个后缀是 s 的真前缀。并且可以发现，从中任意选出两个字符串，较短者既是较长串的前缀，也是后缀。在字符串理论中，我们称较短串为较长串的 border。

更进一步，假设当前最长的一类后缀为 t'，那么**所有**其他的一类后缀都是 t' 的 border。我们也应当在状态中记录最长一类后缀长度 $i = |t'|$。

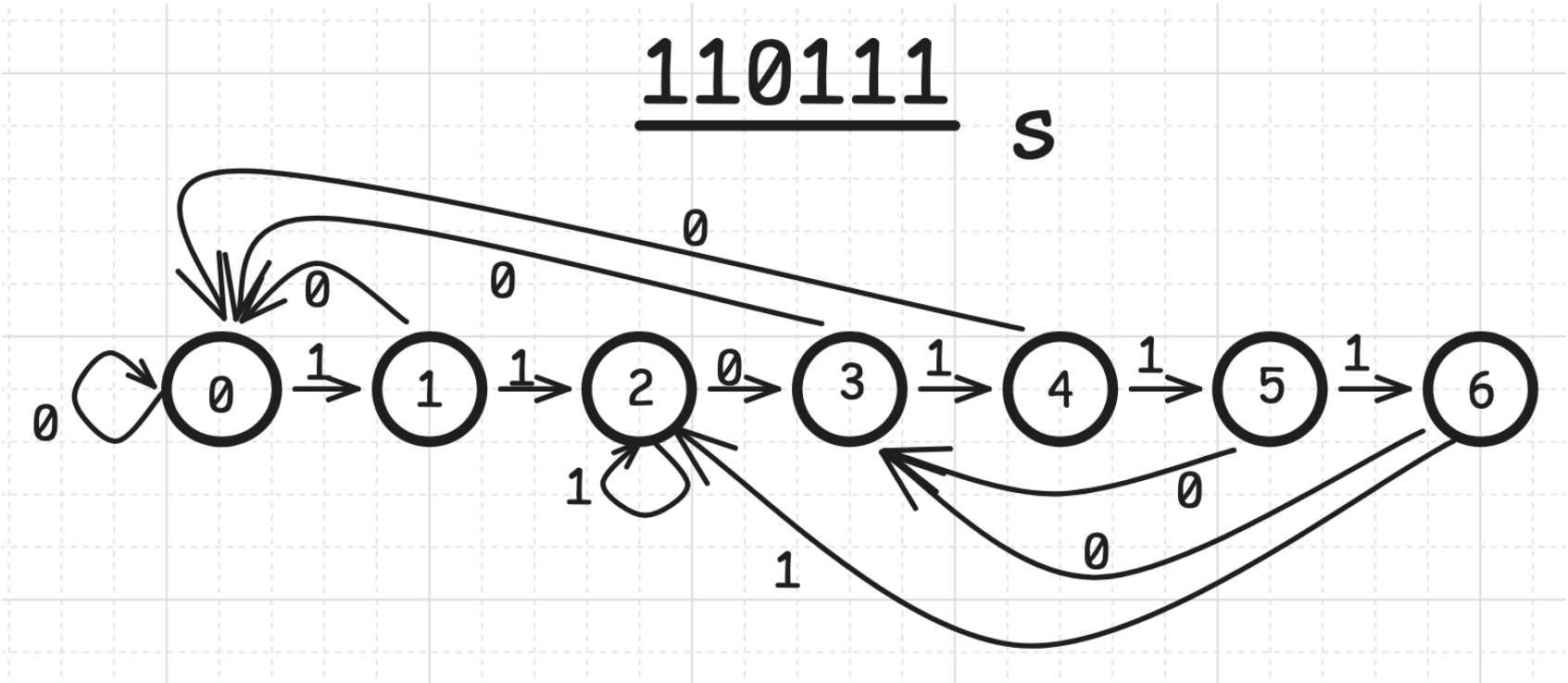


图：t 的所有后缀一类子串

于是一个状态可用五个量 $(|t|, i, b, c, 0/1)$ 刻画，这里的 $0/1$ 表示 s 是否作为子串出现在 t 中。可以发现状态设计上我们不需要记录其他的额外信息。对于每个状态，我们只需记录它是否可达，以及它能从哪些状态转移。

$|t|$ 的转移是容易的。考虑当加入字符时，i 会如何变化。设加入的字符为 x。如果 $x = s_{i+1}$，那么新状态中 $i' = i + 1$。否则，我们需要找到 $s_{[1,i]}$ 的一个 border $s_{[1,j]}$，使得 $s_{j+1} = x$，那么新状态中 $i' = j + 1$。当然如果不存在 j，新状态中 $i' = 0$。

如果我们将转移写成 $i' \leftarrow \delta(i, x)$，那么 δ 会形成一个 KMP-AM（KMP 自动机）。



KMP-AM 会比直接跑“单模式串 ACAM”要好写一点，因为我们可以通过 KMP 求出 fail 指针 π_i（即最长非本身 border）。为了构建边 $\delta(i, 1-s_{i+1})$，我们只需不断令 $i' \leftarrow \pi_{i'}$ 直到 $s_{i'+1} = 1 - s_{i+1}$，然后令 $i' \leftarrow i' + 1$ 即可。

b, c 的转移则可以借助 π 树（即由边 $i \to \pi_i$ 构成的内向有根树）上的信息。当 $x = 0$ 时，可能会使 b 增加。这个增加的量就是“π 树上 i 到根（0）路径上有多少个点 j 满足 $s_{j+1} = 1$”。不妨设这个量为 $dep_{i,1}$。这个量可以通过一次 dfs 求出。当然，更方便的方法是在 KMP 的过程中，直接从 π_i 处继承这个量：

$$dep_{0,1} \leftarrow [s_1 = 1]$$

$$dep_{i,1} \leftarrow dep_{\pi_i,1} + [s_{i+1} = 1]$$

c 的增加量可以分一、二类算。二类的增量就是 b'（指新状态中的 b）。一类的增量就是 π 树上 i 到根路径上点的个数 -1（因为空串不被统计进答案），设为 $dep_{i,0}$：

$$dep_{0,0} \leftarrow 0$$

$$dep_{i,0} \leftarrow dep_{\pi_i,0} + 1$$

总结一下，转移就是：

$$(|t|, i, b, c, 0/1) + 0 \to (|t| + 1, i' = \delta(i, 0), b' = b + dep_{i,1}, c + dep_{i',0} + b', 0/1)$$

$$(|t|, i, b, c, 0/1) + 1 \to (|t| + 1, i' = \delta(i, 1), b, c + dep_{i',0} + b, 0/1)$$

注意一点就是，如果 $i' = n$，那么要将最后一项设为 1，并且 $i' \leftarrow \pi_{i'}$。后者是因为 s 的字典序并不严格小于 s 本身。这个操作要在计算 b', c' 前进行。
粗略估算状态数为 $O(k \times n \times \sqrt{k} \times k) = O(nk^{2.5})$。这里 b 的上界是 $O(\sqrt{k})$ 的。

b 上界的证明.

因为 b 是互不相同的二类后缀的数量，所以它们的长度 $l_1, \ldots, l_b$ 互不相同。这些后缀的所有前缀的字典序都严格 $< s$，所以二类子串的个数至少为 $\sum_{i=1}^{b} l_i \geq \sum_{i=1}^{b} i = \binom{b}{2}$。

于是 $\binom{b}{2} \leq c \leq k$，即 b 的上界是 $O(\sqrt{k})$。 □

现在对于每个状态 $(|t|, i, b, c, 0/1)$，我们记录的是它是否可行。我们肯定要对状态进行优化。可以发现，i, b, c 以及 s 是否出现这四个维度的变化都和 $|t|$ 无关，所以我们可以尝试将 $|t|$ 压掉。正好我们需要求 $|t|$ 的最小值，所以可以直接设 $f_{i,b,c,0/1} = \min |t|$。

这样本质不同的状态数量就是 $O(nk^{1.5})$ 了。因为每次转移 $|t|$ 只会增加 1，所以我们可以使用 bfs 来进行 dp。只要遇到第一个 $(\circ, \circ, k, 1)$ 就可以停止了。

其实甚至不需要维护 $|t|$ 的最小值。因为 bfs 自然地保证了转移的边数最少，所以直接顺着转移路径得到答案即可。最终复杂度 $O(nk^{1.5})$。

~~可以发现最后一维纯属用来增加代码难度。~~

给定长度为 n 的序列 a_i 和长度为 m 的序列 b_j。保证 $n \geq m$。你要对 a_i 进行 $n-m$ 次操作，每次操作为选择下列两项中的一项：

1. 删除 a_i 最左侧的两个元素（即下标最小的两个元素），在 a_i 末尾加入删除的两个数的异或和。
2. 删除 a_i 最右侧的两个元素，在 a_i 的开头加入删除的两个数的异或和。

每次操作会使 a_i 长度 -1。$n-m$ 次操作之后，a_i, b_j 长度相同。问：是否存在恰当的操作方式，使得 $n-m$ 次操作后，序列 a 和 b 相等。若存在，请给构造。

多测，$T \leq 30$，$m \leq n \leq 250$。

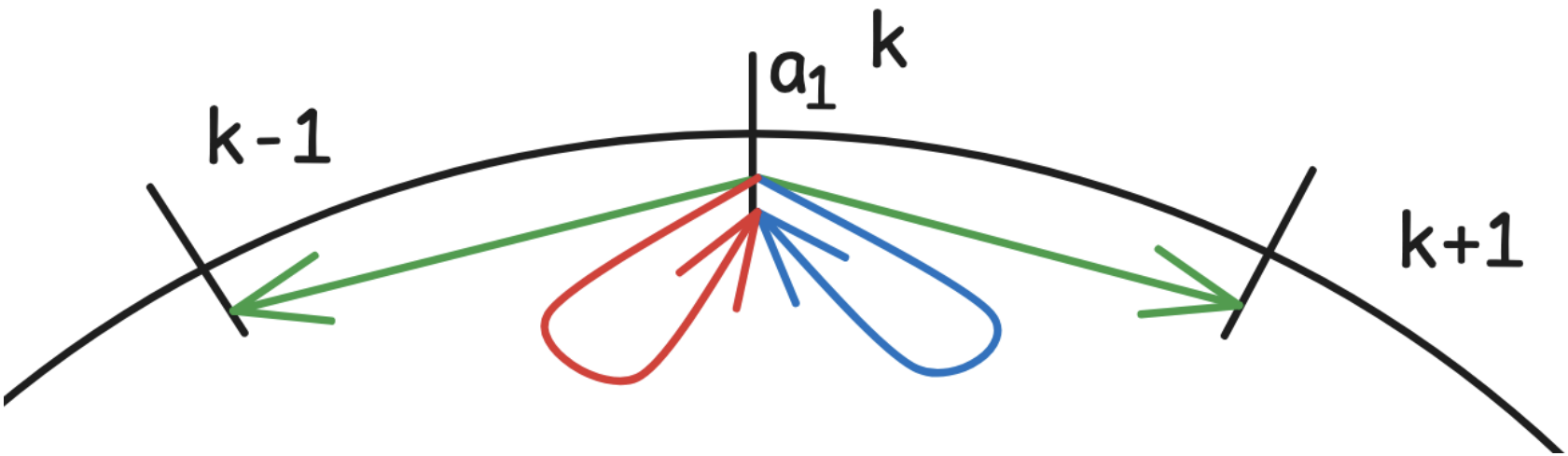
如果将操作视作环上的操作，那么两种操作都是在环上移动两步，并且将跨过的两个数合并为一个数。因此若有解，每个 b_j 都对应了 a 中的一个环上区间。称这样的一个区间为一个**块**。

考虑这样的特殊情形：已经确定了环上区间如何划分，并且初始时 a_1, a_n 不属于一个块。我们要判定这种划分是否合法。

设 bl 表示 a_1 所在的块编号，br 表示 a_n 所在的块编号。则在操作的过程中，有 $bl - br \in \{0, 1\}$（在模 m 意义下）。称其中 $bl = br$ 的状态为**中间状态**，$bl = br + 1$ 的状态为**状态** bl。

我们还要对每个块 k 维护当前的长度 len_k。每次操作都会使一个块的长度 -1，最终要使所有块的长度 $= 1$。

假设初始是状态 k。忽略所有的中间状态，那么每次达到的下一个状态都是 $k \pm 1$ 或者 k 本身。如下图：



如果是回到 k 本身，对 len 的影响是使 len_k 或 len_{k-1} 减去一个 3 的倍数。

证明.

不妨假设走的是蓝色的路径。可以发现，每次操作后，a_1 到 k 块左边界的数量的变化 $\bmod 3$ 都 $= 1$：

- 向右走：将右侧两个数合并为一个移动到左边，左侧数量 $+1$；
- 向左走：将左侧两个数移动到右边然后合并，左侧数量 -2。

□

于是，我们可以将所有 k 不变的操作拆分成若干个使 $len - 3$ 的**基本操作**：“左左右”或“右右左”，取决于当前位于这个块的左侧还是右侧。

对于两条绿色的路径，我们同样可以将其拆成若干个基本操作和只有“左”（或只有“右”）的操作。类似的可以证明，对于一个长度为 l 的块，跨过一条绿色的路径，会使它长度 $\bmod 3$ 的余数变为 $3-(l \bmod 3)$。

在这当中，长度为 3 的倍数的块操作之后还是 3 的倍数。因此，若划分出的块中有一个块长度是 3 的倍数，那么最终它一定不会变为长度 $=1$。即划分不合法。

最终每个块长度都要 $=1$，这至少说明每个块的长度先要变为 $3k+1$。在这之后就可以不断通过基本操作使其变为 1。使一个块 $\bmod 3$ 发生变化的操作就是绿色的路径。

对于一个长度为 $3k+1$ 的块，需要有偶数条绿色路径跨过它；
对于一个长度为 $3k+2$ 的块，需要有奇数条绿色路径跨过它。
特别的，长度为 1 的块不能被任何一种路径跨过（否则就和旁边的块合并了）。

因为一个块从左到右和从右到左的效果是一样的，所以我们可以将跨过的需求视作无向边。

具体的，我们可以对所有块的左端点建一个图。对于一个长度 $\bmod 3$ 余 1 的块 k，它要被绿色路径覆盖偶数次，这相当于在新图中 $k, k+1$ 之间连了偶数条边。对于长度 $\bmod 3$ 余 2 的块，在新图中 $k, k+1$ 之间连奇数条边。

这种条件合法等价于存在一条从最初状态到状态 1 的欧拉路径！

如果我们增加一条从初始状态到状态 1 之间的无向边，则存在欧拉路径等价于连通（指有边的点连通，可以存在一些单点）且每个点度数为偶数。因此上述的“奇数条边”、“偶数条边”可以简化为 1 条边、2 条边。这样每个点度数 ≤ 4。

因为对于每个长度 ≥ 2 的块 k，其两端的点 $k, k+1$ 之间至少有一条边，欧拉路径一定会经过它们，所以可以在第一次靠近这个块时进行基本操作，直到长度为 2 或 4。这就是构造思路。我们只需要找到一个恰当的划分块的方式即可。

在一般情况中，最初状态可能是中间状态，即 a_1 不位于一个块的左边界。

设 a_1 位于第 k 个块。我们可以枚举第一个达到的状态是状态 k 还是状态 $k+1$（即往左达到边界还是往右）。如果选定了方向，类似地可以证明，达到状态时第 k 个块模 3 的余数是固定的。因此存在合法操作是尽量早地达到状态 k 或状态 $k+1$。

我们对 $k = 1, \ldots, m$ 依次决定第 k 个块在 a 中的区间是多少。将 a 复制一遍，然后设状态 (l, r, k) 表示前 k 个块覆盖了 $a_{[l,r]}$。

三个量是不足以描述状态的，我们还需要保证每个点的奇偶性和连通性。一个比较取巧的方法是，只要 $m-1$ 个点的度数都是偶数，那么剩下的点度数也是偶数。所以我们可以不用管 1 的度数。对此，我们还需要补充以下维度：

- c：k 到 $k+1$ 之间的边数（$\in \{0, 1, 2\}$，取决于第 k 个块的长度）；
- a：第 $k+1$ 个点是否能通过 $[1, k]$ 中的点达到点 1。如果为否且 $c \neq 0$，那么 $k+1$ 到 m 都要向右连边；
- s：初始状态是否 $\in [1, k]$。这个维度存在的意义是，我们会给初始状态新增一条边，这个点的奇偶性会翻转。

设 $f(l, r, k, c, a, s)$ 表示这个状态是否可行。转移可以枚举下一个块 $[bl, br]$，它可以从 $r' = bl - 1$ 的状态转移过来。

如果 $[bl, br]$ 包含了 a_n 和 a_{n+1}，那么初始状态要么是 k，要么是 $k+1$。如果选择 $k+1$，那么在转移第 $k+1$ 个块的时候，我们发现 $bl > n$ 且 $s = 0$，这标志着状态 $k+1$ 是初始状态。

点 k 的度数就是上一个状态的 $c'+$ 这个状态的 $c+[k$ 为初始状态 $]$。连通性的转移是容易的。

最终，只要存在一个 $(l, l+n-1, m, \circ, \circ, \circ)$（$\circ$ 表示任意）合法，我们就可以通过转移路径反推出一个合法的块的划分。反之，若不存在这样合法的状态，则不存在构造，报告无解即可。
状态总数是 $O(n^2 m)$ 的（c, a, s 总共 12 种可能，是常数），转移是 $O(n)$ 的，因此总复杂度是 $O(n^3 m)$，无法通过。

可以发现，我们维护的是一个状态是否可行，而且转移中 l 是不变量。因此我们可以用 `bitset` 来维护 l，做到 $O(\frac{n^3 m}{\omega})$，仍然无法通过。

实际上很多转移是无用的。可以证明，除了第一块以外，其他每一块都可以取同余系内最小的块。这个同余系在大部分情况内是模 3 同余系。具体而言就是对于一个左端点 bl，如果有两个 $br < br'$ 都满足区间内的异或和 $= b_k$，而且 $br \equiv br' \pmod 3$，那么我们只用转移 br 而忽略 br'。

证明.

因为异或和相等，所以 $(br, br']$ 的异或和为 0。我们可以将这个区间并到下一个块中。

下一个块可能会因此存在更短的划分，因此继续调整。经过有限次调整之后，可以将第 m 块右侧多余的区间并到第一个块中。此时除了第一个块，其他区间都是右端点最小的区间。 □

对于同时包含了 a_n, a_{n+1} 的块 k，若选择状态 $k+1$ 作为初始状态，则根据贪心，我们要知道 br 的奇偶性。所以跨过 a_n, a_{n+1} 的右端点要按模 6 同余系分类。

这样复杂度就是 $O(\frac{n^3}{\omega} + \frac{n^2 m}{\omega}) = \frac{n^3}{\omega}$ 了。至于记录转移，我们可以对每个 (r, k, c, a, s) 记录所有转移，只要其中有一个满足对应的 $f(l, r, k, c, a, s) = 1$，即可从这个状态递归。可以发现记录的转移总数是 $O(n^2)$ 的。

总复杂度 $O(\frac{n^3}{\omega} + n^2)$。