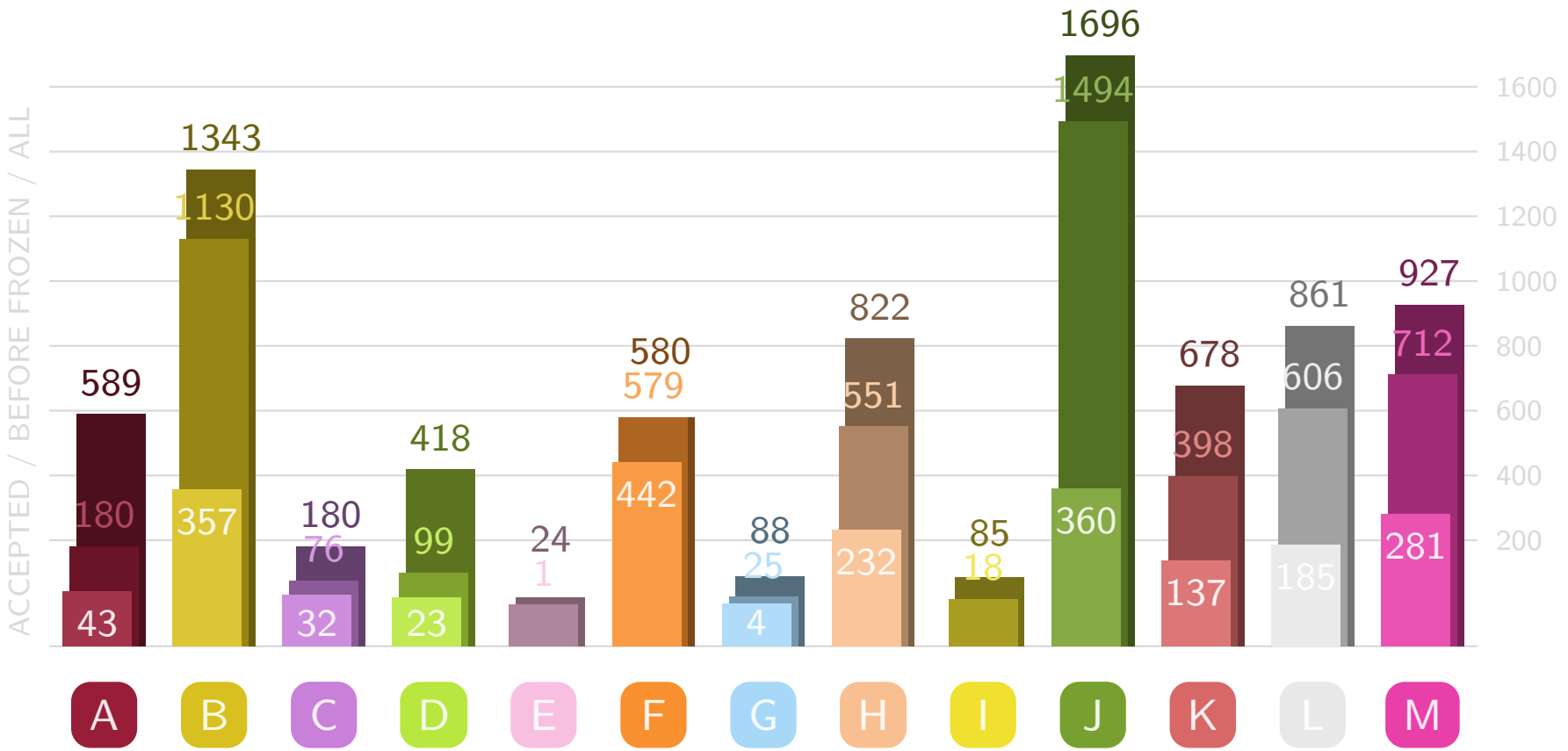
2024 年中国大学生程序设计竞赛
全国邀请赛（郑州）
暨第六届 CCPC 河南省大学生程序设计竞赛
题目解析

2024 年 5 月 12 日

统计


ACCEPTED / BEFORE FROZEN / ALL
1600
1400
1200
1000
800
600
400
200
589
180
43
1343
1130
357
180
76
32
418
99
23
24
1
580
579
442
88
25
4
822
551
232
85
18
1696
1494
360
678
398
137
861
606
185
927
712
281
A
B
C
D
E
F
G
H
I
J
K
L
M

Once In My Life

记 $N = (1234567890 + d) \cdot 10^{\lceil \log_{10} n \rceil}$，则有

$$\lfloor N/n \rfloor \le 2 \cdot 10^9 \cdot \frac{10^{\lceil \log_{10} n \rceil}}{n} \le 2 \cdot 10^{10}$$

并且 $\lfloor N/n \rfloor$ 的前 10 位与 N 相同，因此取 $k = \lfloor N/n \rfloor$ 即可。

单组数据时间复杂度 $O(1)$ 或 $O(\log n)$。

扫雷 1

考虑贪心。假设第 i 轮购买一个地雷探测器所需的扫雷币 c_i 最少，那么第 i 轮及之前轮次获得的扫雷币留到第 i 轮购买最优。剩下的扫雷币与第 $i+1$ 轮开始的 $n-i$ 轮构成子问题，重复贪心直到所有轮次均考虑完即可。

可以发现，可能购买地雷探测器的轮次是 (i, c_i) 这些点的下凸壳，单调栈求出轮次之后模拟即可。时间复杂度 $O(n)$。

中二病也要打比赛

考虑两个相等的数字，根据题目条件，这两个相等的数字中间的所有数字映射后必须是相等的。于是我们可以根据这个性质将整个数列分为若干相等段，例如 $2,\cdots,3,\cdots,2,\cdots,3$ 就是一段相等段。

现在问题变为，为每一段相等段选择一个数字，使得最终序列上升且改变的数字个数最小。先考虑没有相等数字的情形，我们发现只需要关心哪些数字没有改变，因为改变的数字可以任意赋值，所以可以忽略。要使改变的数字的个数最小，等价于在序列中挑选一些数字，让这些数字不变且这些数字单调递增，这是最长上升子序列问题。

中二病也要打比赛

考虑如何扩展到原题目。一种方法是将所有相等段内的数字降序排序，然后再求一次 LIS，求得的答案就是最多的不变个数。LIS 选择出来的那些数字实际上是决定每个相等段最终的数字，又由于每个相等段内降序排序，所以每个相等段内只会选择至多一个数字，LIS 得到的方案是和实际变化的方案一一映射的。时间复杂度 $O(n \log n)$。

没有观察到降序排序的性质也可以通过数据结构和 DP 通过本题。

距离之比

不妨设 $\frac{|x_P-x_Q|}{|y_P-y_Q|}=\tan\theta$，其中 $\theta\in[0,\frac{\pi}{2}]$。有

$$\frac{\|PQ\|_1}{\|PQ\|_2}=\cos\theta+\sin\theta=\sqrt{2}\sin\left(\theta+\frac{\pi}{4}\right)$$

所以 PQ 与 x 轴夹角越接近 $45°$ 或 $135°$，$\frac{\|PQ\|_1}{\|PQ\|_2}$ 越大。

不失一般性地，考虑在 n 个不重合的点中如何求出夹角最接近水平的点对。可以证明，存在一组满足条件的点对，按纵坐标排序后两个点是相邻的。对于原题，我们只需要按照 $x+y$ 与 $x-y$ 排序，取排序后相邻点对计算比值，取最大值即是答案。单组数据时间复杂度 $O(n\log n)$。

保卫城邦

题目大意：初始一棵的树，每次删除一条边并添加一条边，保证图始终为树，操作后对所有点赋点权 val_i，要求对于所有点权为 0 的点 u，与其相邻的所有点 v 满足 $\sum val_v \geq 2$，求最小点权和。

保卫城邦

显然，$val_i \in [0,2]$，于是在固定的一棵树下，钦定树根为 1，我们可以设 $f_{i,j}$ 表示点 i 可以向父亲做出 i 的贡献时，点 i 子树内的最小点权和，其中 $j \in [-2,2]$，当 $j<0$ 时，表示点 i 需要其相邻点再对其进行 $|j|$ 的贡献才合法。设 u 是 v 的父亲，则有如下转移：

$$f_{u,2} = f_{u,2} + \min\{f_{v,2}, f_{v,1}, f_{v,0}, f_{v,-1}, f_{v,-2}\}$$

$$f_{u,1} = f_{u,1} + \min\{f_{v,2}, f_{v,1}, f_{v,0}, f_{v,-1}\}$$

$$f_{u,0} = \min\{f_{u,0} + \min\{f_{v,2}, f_{v,1}, f_{v,0}\}, f_{u,-1} + \min\{f_{v,2}, f_{v,1}\}, f_{u,-2} + f_{v,2}\}$$

$$f_{u,-1} = \min\{f_{u,-1} + f_{v,0}, f_{u,-2} + f_{v,1}\}$$

$$f_{u,-2} = f_{u,-2} + f_{v,0}$$

保卫城邦

设 $dp_{i,j} = \sum_{k=j}^{2} f_{i,k}$，则转移如下：

$$
\begin{aligned}
dp_{u,2} &= dp_{u,2} + dp_{v,-2} \\
dp_{u,1} &= \min\{dp_{u,2} + dp_{v,-2}, dp_{u,1} + dp_{v,-1}\} \\
dp_{u,0} &= \min\{dp_{u,2} + dp_{v,-2}, dp_{u,1} + dp_{v,-1}, dp_{u,0} + dp_{v,0}, \\
&\qquad dp_{u,-1} + dp_{v,1}, dp_{u,-2} + dp_{v,-2}\} \\
dp_{u,-1} &= \min\{dp_{u,2} + dp_{v,-2}, dp_{u,1} + dp_{v,-1}, dp_{u,-1} + dp_{v,0}, dp_{u,-2} + dp_{v,1}\} \\
dp_{u,-2} &= \min\{dp_{u,2} + dp_{v,-2}, dp_{u,1} + dp_{v,-1}, dp_{u,-2} + dp_{v,0}\}
\end{aligned}
$$

由于本题为动态树，需要使用 LCT 实现动态 DP。

保卫城邦

对于一条链，如果我们将矩阵乘法中的乘法改为加法，加法改为取 $\min$，我们可以将转移方程写为矩阵转移形式：

$$\begin{bmatrix} dp_{u,2} & dp_{u,1} & dp_{u,-2} & \infty & \infty \\ dp_{u,2} & dp_{u,1} & dp_{u,-1} & dp_{u,-2} & \infty \\ dp_{u,2} & dp_{u,1} & dp_{u,0} & dp_{u,-1} & dp_{u,-2} \\ dp_{u,2} & dp_{u,1} & \infty & \infty & \infty \\ dp_{u,2} & \infty & \infty & \infty & \infty \end{bmatrix} \begin{bmatrix} dp_{v,-2} \\ dp_{v,-1} \\ dp_{v,0} \\ dp_{v,1} \\ dp_{v,2} \end{bmatrix} = \begin{bmatrix} dp_{u,-2} \\ dp_{u,-1} \\ dp_{u,0} \\ dp_{u,1} \\ dp_{u,2} \end{bmatrix}$$

保卫城邦

可以发现，改写之后的矩阵乘法仍然满足结合律，所以对于一条链上的转移，我们可以将链上所有点的转移矩阵先乘起来，最后乘上 0 个点的 dp 矩阵即可，其中对于 0 个点有 $dp = [0, 0, 0, \infty, \infty]$。

经过以上转化后，我们可以使用 LCT 来维护实链上的矩阵乘积。由于树的形态在不断发生变化，我们需要同时维护相反方向的转移矩阵乘积，以便维护在 LCT 的换根操作里面翻转 Splay 后的信息。

保卫城邦

值得注意的是，对于点 u，其转移矩阵内部的 $dp_{u,i}$ 是只计算虚儿子贡献的值，于是在 LCT 的 access 操作中，u 点切换实儿子时，需要更新其转移矩阵。由于本题的转移较为复杂，无法直接剥离某个儿子的贡献，所以当实儿子切换时，需要重新计算转移矩阵中的值。

保卫城邦

对于点 u，我们可以将儿子们的贡献进行合并，设 $dpson$ 为儿子们的总贡献，于是有：

$$dpson_{-2} = \sum_{v \in son_u} dp_{v,-2} \qquad dpson_{-1} = \sum_{v \in son_u} dp_{v,-1}$$

$$dpson_0 = \sum_{v \in son_u} dp_{v,0}$$

$$dpson_1 = \min_{i \in son_u} \{dp_{i,1} + \sum_{v \in son_u, v \neq i} dp_{v,0}\}$$

$$dpson_2 = \min\{\min_{i \in son_u} \{dp_{i,2} + \sum_{v \in son_u, v \neq i} dp_{v,0}\},$$

$$\min_{i,j \in son_u, i \neq j} \{dp_{i,1} + dp_{j,1} + \sum_{v \in son_u, v \neq i,j} dp_{v,0}\}\}$$

保卫城邦

其中 $dpson_{-2,-1,0}$ 可以轻松维护，$dpson_{1,2}$ 可以整理如下：

$$dpson_1 = \sum_{v \in son_u} dp_{v,0} + \min_{i \in son_u} \{dp_{i,1} - dp_{i,0}\}$$

$$dpson_2 = \sum_{v \in son_u} dp_{v,0} + \min\{\min_{i \in son_u} \{dp_{i,2} - dp_{i,0}\}, \min_{i,j \in son_u, i \neq j} \{dp_{i,1} - dp_{i,0} + dp_{j,1} - dp_{j,0}\}\}$$

于是我们只需要对于点 u 虚儿子们实时维护 $dp_{v,1} - dp_{v,0}$ 的最小值和次小值，$dp_{v,2} - dp_{v,0}$ 的最小值，即可快速合并虚儿子们的 dp 值。利用 set 进行维护会导致单次操作复杂度 $O(\log n)$。

保卫城邦

考虑到 $dp_{v,0} \leq dp_{v,1}$，并且 $dp_{v,0}+1 \geq dp_{v,1}$，可以推出 $0 \leq dp_{v,1}-dp_{v,0} \leq 1$，同理可以推出 $0 \leq dp_{v,2}-dp_{v,0} \leq 2$，所以可以利用桶存储对应值，这样就会将查找最小值和次小值的单次操作复杂度降为 $O(1)$。

需要注意，由于以上 $dp_{v,i}$ 是准确值，所以当 $dp_{v,i}$ 更新时，$dpson$ 也需要进行更新。可以发现，只有当点 v 作为实儿子时，其值才会发生更新。所以当点 u 的实儿子 v 切换为虚儿子时，可以先利用矩阵求出 v 更新后的 dp 值，然后再用其更新 $dpson$。

到此为止，实现的整体复杂度为 $O(125n\log n)$，由于矩阵乘法带来的巨大常数，仍然无法通过此题。

保卫城邦

再次考虑 dp 数组（不包括上面所说的 $dpson$ 数组）的性质，会发现 $dp_{u,-2}$ 与 $dp_{u,-1}$ 同样满足上述推出的 $dp_{u,0}$ 拥有的性质，即：

$$dp_{u,1}-1\le dp_{u,-2},dp_{u,-1},dp_{u,0}\le dp_{u,1}$$

$$dp_{u,2}-2\le dp_{u,-2},dp_{u,-1},dp_{u,0}\le dp_{u,2}$$

同时也易发现 $dp_{u,2}-1\le dp_{u,1}\le dp_{u,2}$。由于 $dp_{u,-2}\le dp_{u,-1}\le dp_{u,0}\le dp_{u,1}\le dp_{u,2}$，在这些条件的限制下，$dp$ 数组的差分数组只有 8 种可能，如果把差分数组相同的 dp 数组归为一类，那么我们只有 8 种 dp 数组。

保卫城邦

于是我们还可以猜测，实际上转移矩阵的种类数也不是很多。为了方便研究，我们可以在不影响结果的情况下将矩阵转移式改写如下：

$$\begin{bmatrix} dp_{u,2} & dp_{u,1} & dp_{u,-2} & dp_{u,-2} & dp_{u,-2} \\ dp_{u,2} & dp_{u,1} & dp_{u,-1} & dp_{u,-2} & dp_{u,-2} \\ dp_{u,2} & dp_{u,1} & dp_{u,0} & dp_{u,-1} & dp_{u,-2} \\ dp_{u,2} & dp_{u,1} & dp_{u,1} & dp_{u,1} & dp_{u,1} \\ dp_{u,2} & dp_{u,2} & dp_{u,2} & dp_{u,2} & dp_{u,2} \end{bmatrix} \begin{bmatrix} dp_{v,-2} \\ dp_{v,-1} \\ dp_{v,0} \\ dp_{v,1} \\ dp_{v,2} \end{bmatrix} = \begin{bmatrix} dp_{u,-2} \\ dp_{u,-1} \\ dp_{u,0} \\ dp_{u,1} \\ dp_{u,2} \end{bmatrix}$$

保卫城邦

经过改写之后，我们可以证明，对于所有的转移矩阵（包括维护时转移矩阵相乘所得到的转移矩阵），都满足以下性质：

1. 对于每一行 row，从左到右看，其值单调递减，并且满足 $row_0 - row_1 \leq 1$，$row_0 - 2 \leq row_2, row_3, row_4 \leq row_0$，$row_1 - 1 \leq row_2, row_3, row_4 \leq row_1$。
2. 对于每一列 col，从上到下看，其值单调递增，并且满足 $col_4 - col_3 \leq 1$，$col_4 - 2 \leq col_0, col_1, col_2 \leq col_4$，$col_3 - 1 \leq col_0, col_1, col_2 \leq col_3$。

即每一行与每一列都满足与 dp 数组相同的限制，行和列的种类都只有 8 种。

保卫城邦

大致证明如下：首先，对于所有由单个 dp 数组所得到的初始转移矩阵，显然满足性质。

对于两个满足性质的矩阵相乘，首先考虑一个矩阵与一个列向量相乘，而这又可以先考虑矩阵的一个列向量与列向量中的一个值相乘：

$$\begin{bmatrix} f_{0,0} \\ f_{1,0} \\ f_{2,0} \\ f_{3,0} \\ f_{4,0} \end{bmatrix} \begin{bmatrix} c_0 \end{bmatrix} = \begin{bmatrix} f_{0,0} + c_0 \\ f_{1,0} + c_0 \\ f_{2,0} + c_0 \\ f_{3,0} + c_0 \\ f_{4,0} + c_0 \end{bmatrix}$$

保卫城邦

其所得到的列向量满足性质。而矩阵与列向量的乘积所得到的列向量，就是五个列向量对应位置取最小值所得到的列向量。设 $g_{i,j} = f_{i,j} + c_j$，则：

$$\begin{bmatrix} f_{0,0} & f_{0,1} & f_{0,2} & f_{0,3} & f_{0,4} \\ f_{1,0} & f_{1,1} & f_{1,2} & f_{1,3} & f_{1,4} \\ f_{2,0} & f_{2,1} & f_{2,2} & f_{2,3} & f_{2,4} \\ f_{3,0} & f_{3,1} & f_{3,2} & f_{3,3} & f_{3,4} \\ f_{4,0} & f_{4,1} & f_{4,2} & f_{4,3} & f_{4,4} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} = \begin{bmatrix} \min_{i=0}^{4} g_{0,i} \\ \min_{i=0}^{4} g_{1,i} \\ \min_{i=0}^{4} g_{2,i} \\ \min_{i=0}^{4} g_{3,i} \\ \min_{i=0}^{4} g_{4,i} \end{bmatrix}$$

保卫城邦

不妨设 $\min_{i=0}^{4} g_{0,i} = g_{0,0}$，那么 $g_{0,0} \le \min_{i=0}^{4} g_{j,i} \le g_{j,0}$。由于 $\{g_{j,0}\}$ 满足性质，所以该列向量也一定满足性质。由此可以得知，所有的列都满足性质。同理可得，所有的行也都满足性质。所以，两矩阵相乘所得到的矩阵满足性质。

综上所述，所有转移矩阵都满足上述性质。

在以上性质的限制下，合法矩阵种类数为 5586 个。打表后会发现，实际可能出现的转移矩阵只有 1255 种。于是我们可以打表预处理出所有矩阵相乘的结果，记录不同矩阵相乘后结果的种类与最小值的变化。然而直接利用矩阵乘进行预处理，会导致复杂度来到 2×10^8 级别，对于程序来说仍然难以接受。

保卫城邦

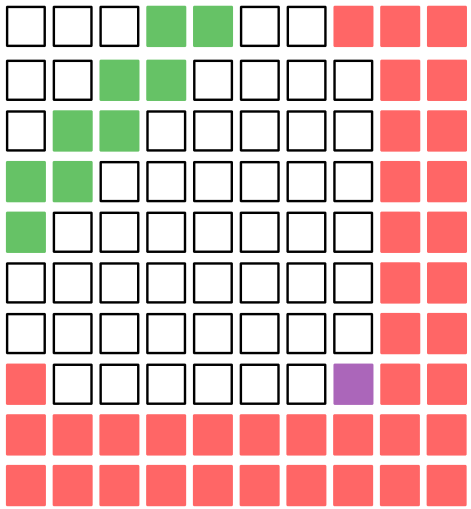
我们可以先处理向量与向量之间相乘的结果，再处理向量与矩阵之间相乘的结果，最后预处理出矩阵与矩阵相乘的结果，这样下来预处理的复杂度会变得可以接受。在预处理后，矩阵乘法的复杂度变为 $O(1)$，大幅减少了矩阵乘法带来的常数，即可通过此题。总体时间复杂度为 $O(S^2 + n \log n)$，其中 S 为矩阵种类数，其值为 1255。

优秀字符串

签到题，按照题意实现代码即可。

扫雷 2

构造方法很多，以下是一种可行的方法。



先把多余的雷放在红色部分，放到放不完一层为止。剩下的放绿色蛇形，蛇形的个数可以取到所有奇数。最后如果剩一个就放在紫色位置。

随机栈

取出所有满足 $0 \le a_i \le n$ 的元素，并将它们排序，记为 $\{b_1, b_2, \ldots, b_n\}$。显然，最终产生的序列应该是 b。

我们可以直接按顺序模拟这个过程。设初始概率为 1：

- 若 $0 \le a_i \le n$，那么将该元素插入多重集中。
- 否则，我们已经知道下一个需要取出的元素。
  - 若该元素不存在于多重集中，答案为 0。此时可直接输出 0 并返回。
  - 否则，将答案乘上该元素的出现次数/集合总大小。

由于需要排序，时间复杂度为 $O(n \log n)$。本题也有时间复杂度 $O(n)$ 的算法，这里留给读者自己思考。

378QAQ 和字符串

首先考虑从 1 到 $\frac{n}{2}$ 枚举 p，对于给定的 p，考虑字符串 s 长度为 $n-p$ 的前缀 t_1 和后缀 t_2 进行匹配。后缀数组支持 $O(n\log n)$ 预处理并 $O(1)$ 查询一个字符串两个子串的最长公共前缀 (LCP)，而 LCP 的下一个位置即为失配位置。根据后缀数组我们可以不断的查找 t_1 和 t_2 的失配位置，当 t_1 和 t_2 出现了超过 $2k$ 次失配我们停止匹配，因为一次修改至多影响两个失配位置。该部分的时间复杂度为 $O(n\log n+nk)$。

378QAQ 和字符串

如果失配次数不超过 $2k$ 次，我们考虑如何修改 k 个位置使得字符串满足题意。如果任意两个位置 i 和 j 满足 $i \equiv j(mod\ p)$，假设有 $j = i + mp$，则对于修改后的字符串有 $s_i = s_{i+p} = s_{i+mp} = s_j = s_{j+p}$，此时我们将 i 和 j 视为等价位置。对于一个失配位置，我们可以统计所有和它等价的位置中出现的字符次数，从贪心的角度考虑，我们找出出现次数最多的字符并将其他位置修改为该字符。如果修改总次数小于 k，则答案为 Yes，否则为 No。该部分的复杂度为 $O(k\sum_{p=1}^{\frac{n}{2}} \frac{n}{p})$。

378QAQ 和字符串

此时总复杂度为 $O(nk\log n)$，无法通过。考虑如何优化，假设存在 p_1 $(2p_1 \le \frac{n}{2})$ 满足要求，可以发现此时有 $s_i = s_{i+p_1} = s_{i+2p_1}$，此时 $p_2 = 2p_1$ 也满足要求。因此枚举范围可以缩小到从 $\frac{n}{4}$ 到 $\frac{n}{2}$，此时总复杂度为 $O(n\log n + nk)$。

排列与合数

注意到以 0、2、4、5、6、8 之一结尾的多位整数都是合数，根据鸽巢原理，给定的整数 n 中至少有一个数位在 0、2、4、5、6、8 之中。将满足条件的数位放至整数末尾即可得到满足要求的答案。单组数据时间复杂度 $O(1)$。

作为简单题，枚举全排列再检查也可以通过本题。

树上问题

对于一对相邻的节点 u 和 v，如果有 $a_u \geq \frac{a_v}{2}$，此时节点 u 可以作为节点 v 的父亲节点，此时在两个节点之间加入一条从 u 到 v 的有向边。$a_v < \frac{a_u}{2}$ 时同理，节点 v 可以作为节点 u 的父亲节点，此时在两个节点之间加入一条从 v 到 u 的有向边。

树上问题

如果一对相邻的节点 u 和 v 都有有向边从自己连向对方，我们可以将其缩成一个点，因为无论哪个节点作为根，节点 u 和 v 都可以作为对方的父亲节点。我们将所有复合条件的点对缩点之后，就得到了一个有向无环图（并且如果将图中的边改为无向边，该图是一棵树）。如果存在美丽节点，则在该有向无环图仅存在一个节点入度为 0，其余节点入度均为 1，此时将入度为 0 的点作为根时，其余节点都满足点权要求。

最后答案即为该节点包含的原始节点数。时间复杂度 $O(m\alpha(n)+n)$。

Toxel 与 PCPC II

显然，每次运行代码时一定会停在某个有 bug 的行。设 dp_i 表示 debug 前 i 行所需的最小时间，那么我们有转移方程：

$$dp_i = \min_{1 \le j \le i} \left(dp_j + a_i + (i-j)^4 \right)$$

这个 dp 的时间复杂度是 $O(n^2)$，不能通过本题。但是我们可以注意到，debug 所需的时间为 bug 的四次方，代价很大，所以一次只会进行少量的 debug。

Toxel 与 PCPC II

不妨设一次 debug 了 x 个 bug。另一种方案是考虑将它分为两次，每次 debug $x/2$ 个 bug。这样额外带来的运行时间为至多 n。求解不等式 $x^4 \geq 2 \cdot (x/2)^4 + n$，可得 $x \geq \sqrt[4]{8n/7}$。即当 x 大于该值时，拆成两部分 $x/2$ 分别 debug 更优。那么，我们每次求 dp 时只需要向前遍历 $O(n^{1/4})$ bug 即可。

时间复杂度 $O(n^{5/4})$。

有效算法

考虑二分答案。设答案为 k，则第 i 个数能取得的范围为 $[a_i - k \times b_i, a_i + k \times b_i]$，有解当且仅当这 n 个取值范围区间有交。判断是否有交只需要判断左端点的最大值是否不大于右端点的最小值即可。时间复杂度 $O(n \log a)$。