

Problem A. Well Rested

判断 01 串中有没有连续的六个 1，也就是判断输入串中有没有“111111”这个子串。可以扫一遍这个串，一边扫一边记录当前连续的 1 的个数，也可以直接用 C 语言中的 `strstr` 函数，C++ 语言中的 `string::find` 函数，或者 Python 语言中的 `find` 函数等查找子串。

Problem B. 毕业合照

直接 $O(n^2)$ 枚举即可。

如果两个人向对方走来，或者方向相同且后面的人可以追上前面的人，都可以产生一张合照。

Problem C. 咒语计数

提供以下两种做法：

第一种是动态规划，考虑用 $dp_{i,j}$ 表示考虑到 s 的第 i 个位置，串“cvbb”匹配到了第 j 个字符的答案，那么有两个转移： s_i 不匹配“cvbb”中的字符，此时 $dp_{i,j}$ 从 $dp_{i-1,j}$ 转移；如果 s_i 与“cvbb”的第 j 个字符相等，则 $dp_{i,j}$ 还可以从 $dp_{i-1,j-1}$ 转移。初值 $dp_{0,0} = 1$ ，答案是 $dp_{n,4}$ 。时间复杂度 $O(n)$ 。

第二种做法是直接计数。枚举“cvbb”中的‘v’在 s 中的位置 i ，假设 s 在 i 位置之前有 pre_c_i 个‘c’，在 i 位置之后有 suf_b_i 个‘b’，那么枚举‘v’过后只需要在前面的‘c’中选一个，在后面的‘b’中选两个。因此答案是 $\sum_{i=1}^n pre_c_i \times \frac{pre_b_i(pre_b_i-1)}{2}$ 。 pre_c_i 和 suf_b_i 都可以提前扫一遍 s 算出。时间复杂度 $O(n)$ 。

这道题本来设置的 n 的上界是 2×10^5 ，这样的话答案最大情况下会超出 `long long` 的范围，但是恰好又在 `unsigned long long` 的范围内。后来在其他出题人的建议下没有这么做。

Problem D. 无声之歌

考虑令 $s_0 = 0, s_i = \sum_{j=1}^i a_j$ 表示数列 a 的前缀和，那么 $sum(l, r) = s_r - s_{l-1}$ 。首先考虑计算最大非空子段和。我们可以枚举 r ，然后考虑所有小于 r 的 l 如下

$$MaxSum = \max_{1 \leq l \leq r \leq n} s_r - s_{l-1} = \max_{0 \leq l < r \leq n} s_r - s_l = \max_r (s_r - \min_{0 \leq l < r} s_l)$$

扫一遍 r 同时维护 $\min_{l < r} s_l$ 即可。然后考虑求次大，做法与上面基本相同，但是从小到大扫 r 的时候要维护满足 $l < r$ 最小的和次小的 s_l ，更新答案的时候，确保 $s_r - s_l \neq MaxSum$ 才更新。时间复杂度 $O(n)$ 。

Problem E. Serval 的树

可以发现，每次选择一个仍有边相连的点操作不会更劣，因此存在一种最优操作序列，使得操作后图中无边。注意到若操作后图中无边，则图中结点数为树的结点数与操作次数之差，最小化操作次数与操作后图中剩余边数之和只需最大化操作后图中结点数，这即是树的最大独立集。动态规划或是贪心均可解决，时间复杂度 $O(n)$ 。

Problem F. 火柴棒等式

可能的火柴总数相同的火柴棒等式只有不到 6×10^4 种，直接对于每一种和输入时总火柴棒相同的等式判断即可。DFS 或者数位 DP 也可以通过，没有构造数据去卡任何做法。

Problem G. 石子配对

忽略每堆石子数量的问题。考虑若干个石子，如何选取才能最大化和谐度。

首先可以先将石子大小进行预处理，将其控制在 $[0, k - 1]$ 范围内。此时和谐度的计算公式变为 $v_i + v_j - [v_i + v_j \geq k] \times k$ 。问题转化为如何匹配才能使得 $v_i + v_j \geq k$ 的匹配对数最少。

我们将石子按照大小进行排序，从大到小考虑每个石子的决策。

- 当目前最大的石子 j ，与其余任意一个石子 i 匹配后，都有 $v_i + v_j \geq k$ 的情况。可以发现，将最大的石子与次大的石子进行匹配一定不劣于其余选择。
- 否则，可以匹配且不产生代价的石子一定是该石子序列的一个前缀，我们可以称其为一个“好的匹配”。前缀中的每一个石子 i 与最大的石子匹配，皆有 $v_i + v_j < k$ 。换言之，它们都可以用与目前任意石子匹配而不对答案产生任何的代价。此时，前缀中的石子并不存在什么本质不同，我们可以令最大的石子与其中任何一个进行匹配。

综上所述，采取贪心策略：从大到小对石子进行决策，如果当前不能找到“好的匹配”，则令它与次大石子进行匹配。否则，任选石子与它进行“好的匹配”。

对于原题中，每堆石子有不同数量的问题，可以使用双指针的方法进行解决。只需不停地令最大的石子堆与最小的石子堆进行“好的匹配”，或者在最大的石子堆内部匹配，计算匹配的对数即可。时间复杂度 $O(m \log m)$ 。

Problem H. 矩阵乘法

考虑枚举第一个矩阵的四个数的值和第二个矩阵四个数的位数，发现第二个矩阵的值可以由四元一次方程唯一确定。

列出方程可以发现该四元一次方程是两个不相交的二元一次方程，直接手解即可。

时限给的很宽，不需要剪枝即可通过。

Problem I. 坏掉的计数器

注意到一些显然的结论：

- 最坏情况下，所有初始不亮的数码管均坏掉了；
- 一位数无法确定，当且仅当所有数码管都坏掉了；
- 一位数能直接确定，当且仅当其只能为 8；
- 答案为 -1 ，当且仅当最高位无法确定；

实际上，确定一个数，就是让所有和它不同的数的数码管图案不完全相同。也就是说，我们需要求 $\max_{0 \leq x < y < 10^n} f(x, y)$ ，其中 $f(x, y)$ 为 x 和 y 至少需要多少次 $+1$ 操作才能让图案第一次不同。

注意到 x 和 y 应该恰有一位不相同，分类讨论后发现只有两种情况：

$x = 888X****, y = 888Y****$ 或 $x = 888X000***, y = 888X001***$ ，其中 8 为直接确定位，0 为不确定位，* 为当前位最小可能值。预处理出一位的情况之后，分别计算之后取最大值即可得到答案。（大

部分 WA 的提交都是因为没有考虑到第二种情况，在 2/1000000/0000000 这组数据上出现错误，这组数据的答案是 59 而不是 50）

有一些边界情况，例如是否有 8 等等，如果一直 WA 可以写一个对拍，三位数即可覆盖所有情况。

这道题是问号同学精心准备的题目，题意简洁，不需要任何算法却有区域赛银牌难度。如果初学者能独立通过此题，问号同学强烈推荐你参加 ACM 竞赛。

Problem J. while (1) replace;

考虑把 ‘a’、‘b’、‘c’ 看成 11 进制下的数，‘a’ 为 1，‘b’ 为 11，‘c’ 为 11^2 。这样处理之后就非常容易统计答案了，我们可以得到如下解决方案：

- $c \rightarrow b \times 11$
- $b \rightarrow a \times 11$
- $a \times 121 \rightarrow 1$
- $a \times 11 \rightarrow b$
- $11 \rightarrow 1$
- $bb \rightarrow b$
- $aa \rightarrow a$
- $a \rightarrow 1$
- $b \rightarrow 1$
- $111 \rightarrow 3$
- $11 \rightarrow 2$

只使用了 11 次操作，其中 $c \rightarrow b \times 11$ 代表 `replace("c", "bbbbbbbbbbb")`。

Problem K. Power Shift

对于每个位置，注意到一个平方操作可以和之后的开方操作抵消掉。最终每个位置的操作序列一定等价于先进行若干次开方操作，再进行若干次平方操作。考虑用线段树来维护。在进行 pushdown 的时候，被下传的标记一定是晚于子节点的标记的，直接合并在后面即可。时间复杂度 $O(q \log n)$ 。

Problem L. 数列计数

注意到除了 $F(l, r) = F(u, v)$ 的情况以外，对于 $F(l, r)$ 和 $F(u, v)$ ，必然有以下二者之一：

- $F(l, r) < F(u, v)$, $F(u, v) > F(l, r)$
- $F(l, r) > F(u, v)$, $F(u, v) < F(l, r)$

于是 $F(l, r) < F(u, v)$ 的数量即为不等号成立的有序对数量的一半。不等号成立数量为有序对总数与等号成立数量的差值，故只需统计等号成立的数量。

如果给定的数列中没有 0，即是求一个字符串中有多少对相同子串。可以先后缀数组排序再单调栈解决，也可以后缀自动机解决。而存在 0 时需要在计数时注意计入，后缀自动机需要利用转移边信息。后缀数组时间复杂度 $O(n \log n)$ ，后缀自动机时间复杂度 $O(n)$ 。

Problem M. Easy XOR problem

这题有经典 $O(n^2m)$ 和 $O(nm^2)$ 做法, 通过数据分治即可得到复杂度为 $O(n^{1.5}m^{1.5})$ 的做法。具体地说, $O(n^2m)$ 做法按题意枚举即可, $O(nm^2)$ 做法考虑枚举两个位 i, j 的贡献, 若 $i = j$ 则系数为 1, 若 $i \neq j$ 则系数为 2, 这是因为 $x^2 = (\sum_p x_p)^2 = \sum_p x_p^2 + 2 \sum_{p \neq q} x_p x_q$ 。

考虑如何优化这个做法。对于 $O(n^2m)$ 的做法我们可以简单压位得到 $O(\frac{n^2m}{w})$ 的做法。考虑如何优化 $O(nm^2)$ 的做法, 我们发现在计算上面那个式子的时候, 我们只关心 i, j 位是 00、01、10、11 这几种情况的出现次数, 于是我们就可以通过压位求出这几种情况出现次数, 时间复杂度为 $O(\frac{nm^2}{w})$ 。

总复杂度为 $O(\frac{n^{1.5}m^{1.5}}{w})$, 其中 $w = 64$ 。

Problem N. Purple Shop

首先先通过 n 次询问确定每种颜色各出现了多少次, 然后接下来分类讨论:

1. 如果有某种颜色完全没有出现过, 那么我们可以对其余的颜色进行二分查找其所在位置, 更具体的来说就是: 对于颜色 c , 已知其在衣饰区间 $[l, r]$ 内出现过 s 次, 猜测区间 $[l, mid]$ 的颜色均为 c , 其余均猜测为未出现的颜色, 如果返回值 x 不为 0, 那么对 $[l, mid]$ 内递归, 同时如果 $s - x \neq 0$, 那么递归查询区间 $[mid + 1, r]$ 。如此即可确定所有颜色出现的位置。
2. 如果没有颜色完全未出现过, 那么代表每个颜色各出现了一次。我们可以通过 n 次询问查询出出现在 $[1, mid]$ 的颜色和 $[mid + 1, n]$ 的颜色, 并对左右区间分别进行递归。更具体的来说就是: 每次询问时, 钦定区间 $[1, mid]$ 为颜色 1, 其余部分遍历 $2 \sim n$, 如果返回值为 1, 说明该颜色与颜色 1 同在左区间或右区间, 其余颜色返回值如果为 0, 则说明颜色 1 在右区间, 否则在左区间。递归到子区间后以此类推, 由此即可确定每个颜色所在位置。

显然, 以上两种情况的做法都需要 $O(n \log n)$ 级别次询问, 然而本题对询问次数限制较为严格, 同为 $O(n \log n)$ 级别次询问, 可能仍然无法通过该题目。

Problem O. Toxel 与字符串匹配

首先考虑对问题做一些变形。我们可以求每种偏移下匹配的元素数量, 最终只需用每种偏移重叠的长度减去它即可得到答案。这样带来的好处是, 我们可以对每种元素分别计算答案。

对于某种元素, 最暴力的做法是分别枚举它在 a, b 中的出现位置, 然后写一个二重循环计算每对位置的距离, 将它对应的偏移量的答案加一。当这个元素在 a, b 中都出现很多次时, 这样做复杂度就很高了。有没有更好的办法呢? 注意到我们要对 a 中的位置减去 b 中的位置做一个计数, 这正好是一个卷积形式。我们可以用 NTT 来加速, 复杂度为 $O(n \log n)$ 。多项式的构造非常直接, 请读者自行思考。

到这里, 本题的解法已经呼之欲出了。我们可以采用根号分块的思想, 当该元素出现次数较多时, 使用 NTT, 否则使用暴力。考虑分析一下复杂度。为了方便这里假设 $n = m$ 。对于暴力做法, 根据均值不等式, 每种元素在两边数量相等时取到最坏复杂度, 而 NTT 对于每种元素的复杂度固定为 $O(n \log n)$ 。假设我们设定的块阈值大小为 T , 那么就有 $T^2 > n \log n$ 时, 我们使用 NTT。 $T > \sqrt{n \log n}$ 的块最多只有 $\sqrt{\frac{n}{\log n}}$ 个, NTT 的总复杂度为 $O(n \sqrt{n \log n})$ 。对于暴力的块, 最坏情况下考虑 a 中每个元素都要枚举 b 中对应的块内所有元素, 总复杂度也为 $O(n \sqrt{n \log n})$ 。

注意 NTT 常数极大, 这个阈值 T 事实上要比 $\sqrt{n \log n}$ 更大一些。标程在 $T \approx 1.4 \times 10^5$ 处达到了最快的运行速度。