MX-S5 讲解

Inaba_Meguru

FeOI、TFOI、KDOI 联合命题组

2024/11/9

视频题解

https://www.bilibili.com/video/BV1juDrY1ECb/

题意简述

给定一个长度为 n 的 01 字符串 $T = S^{\infty}$。
一个人在 T 上移动。他的一次移动为：设当前下标为 x，则跳到 $[x+1, x+m]$ 中最后一个为 1 的位置，如果区间中不存在 1 则移动至 $x+1$。有 q 次询问，每次询问给定 x, y，求从 x 开始移动 y 次后的下标对 10^9+7 取模的值。
对于 100% 的数据，$1 \le n, q \le 2 \times 10^5$，$1 \le m, x \le 10^9$，$1 \le y \le 10^{18}$。
Author: Supor_Shoep

算法 1

我们称 T 中的每 n 个字符为一个**周期**，那么显然第 i 个字符是属于第 $\lceil \frac{i}{n} \rceil$ 个周期的。记 b_i 表示 i 在它所在周期中的**相对位置**，则有 $b_i = i - (\lceil \frac{i}{n} \rceil - 1) \times n$。

对于每次询问，我们就暴力移动 y 次，每次从 $x + m$ 开始向前枚举，若当前枚举到的下标 i 满足 T_{b_i} 为 1，则让 $x \leftarrow i$。

时间复杂度最劣 $\mathcal{O}(m \times q \times y)$，期望得分 10。

算法 2

对于特殊性质 A，若 S 为全 0 串，那么每次移动必然是 $x \leftarrow x+1$，输出 $(x+y) \bmod (10^9+7)$即可。否则每次移动必然是 $x \leftarrow x+m$，输出 $(x+y \times m) \bmod (10^9+7)$ 即可。
时间复杂度 $\mathcal{O}(n+q)$，期望得分 10。

算法 3

我们先从 $f(i,1)$ 的求法入手。设 pre_i 表示 T 串的 $1\sim i$ 位中**最后一位** 1 的位置，则题目就是让我们求 $\max\{i+1,pre_{i+m}\}$。对于 $1\sim n$ 的 pre_i 我们可以直接预处理；对于其余的 i（$i>n$），我们发现有 $pre_i=pre_{i-n}+n$，稍微扩展一下就是 $pre_i=pre_{b_i+n}+(i-b_i)-n$。

因此我们就可以只预处理 $[1,n]$ 的 pre_i，其余的部分自然都能够表示出来。那么现在 $f(x,1)$ 就是可以直接 $\mathcal{O}(1)$ 询问的了。

此算法可以通过 $y=1$ 的测试点。时间复杂度 $O(n+q)$，期望得分 15。

算法 4

可以发现当我们移动了若干次之后，我们当前下标 i 对应的 b_i 会进入一个循环节。考虑在每一组询问中记录一个标记数组 vis_i，对于第 t 次移动后到达的下标 i 让 $vis_{b_i} \leftarrow t$，如果在一次移动中，我们发现 vis_{b_i} 已经有值，那么就代表第 vis_{b_i} 次移动到第 t 次移动形成了一个循环节。于是我们就把剩余的移动次数 y 对 $t - vis_{b_i}$ 取模，计算我们还需要走多少次**完整的循环节**，对于取模之后的 y，我们就暴力处理完这 y 次移动即可。

每次找到一个循环节的最劣时间复杂度为 $\mathcal{O}(n)$，因此最劣的总时间复杂度位 $\mathcal{O}(nq)$，但是这个算法可以必然地通过特殊性质 B 和特殊性质 C。期望得分 55，实现够好应该可以获得更高分。

算法 5

现在考虑一个简单的性质：$f(i+n,j)=f(i,j)+n$。
因此对于 $f(i,j)$，我们可以简单地看为求 $f(b_i,j)$，最后再加上 $(\lceil\frac{i}{n}\rceil-1)\times n$ 即可。

算法 5

那么怎么回答移动次数极大的询问呢？考虑**倍增**，设 $f_{i,j}$ 表示从 i 开始移动 2^j 次之后到达的下标在其所在周期中的**相对位置**，$g_{i,j}$ 表示从 i 开始移动 2^j 次之后到达的位置**之前的完整周期个数**。举个例子解释一下：下面的串中，每一个括弧为一个周期，标记为蓝色的 1 所在的相对位置为 3，它之前的完成周期个数也是 3，即在它之前的括弧数量。

$$\underbrace{1011\ldots1101}\ \underbrace{1011\ldots1101}\ \underbrace{1011\ldots1101}\ \underbrace{1011\ldots1101}$$

算法 5

根据基本的倍增转移方程，我们有：

$$\begin{cases} f_{i,j} = f_{f_{i,j-1},j-1} \\ g_{i,j} = g_{i,j-1} + g_{f_{i,j-1},j-1} \end{cases}$$

之后对于每次询问，我们把 y 拆成二进制，按照二进制中的每个 1 进行倍增即可。时间复杂度 $\mathcal{O}(n \log V)$，足以通过本题，期望得分 100。

题意简述

你要买 n 个物品，第 i 个物品原价 a_i，折扣价 b_i。
你还有 m 个优惠券，第 i 个优惠券形如**原价**满 w_i 减 v_i。
对于第 i 个物品，你可以选择以下三种购买方式之一：
1. 使用原价 a_i 购买。 2. 使用折扣价 b_i 购买。 3. 选择一个优惠券 j，要求满足 $a_i \geq w_j$，使用优惠券 j，以 $a_i - v_j$ 的价格购买。注意每个优惠券 j 只能被最多一个 i 使用。
求购买所有物品最少用钱。
Author: lizhous

算法 1

考虑特殊性质 A。
把物品按 a 升序排序后，每个优惠券可以被使用在一个后缀的物品上。
由于选择按折扣价购买不会省钱，所以我们要尽可能用优惠券省更多的钱。
从前往后扫物品，每经过一个物品会有一些优惠券变得可以被使用，每次在当前可以使用且未使用的优惠券中找到省钱最多的使用即可。
具体来说可以用优先队列维护当前所有可以使用且未使用的优惠券的 b 值，每次直接找到最大的使用。
时间复杂度 $O(n)$，期望得分 20。

算法 2

考虑特殊性质 B。
首先可以把折扣价改写为可以选择优惠 $c_i = a_i - b_i$ 钱。现在每个物品我们先选择折扣价，然后用优惠券来省钱。
按任意顺序扫描物品，对于一个物品，有三种选择：

- 用当前没使用过的省钱最多的优惠券。
- 把已经使用过的优惠券换到这个物品。而被拿走优惠券的物品改为使用折扣价。
- 用优惠价。

算法 2

找到最优的使用。
说明一下 2 为什么被拿走优惠券的物品改为使用折扣价。首先假如从 j 换到 i，会让答案变优一定是 $c_j \geq c_i$，那么如果 j 改为使用其他优惠券，那么和这张优惠券直接用在 i 省的钱是一样的。
寻找 2 需要优化，若从物品 j 换一个可以优惠 v 钱的优惠券到物品 i，j 物品多 $v - c_j$ 钱，i 物品少 $v - c_i$ 钱，所以答案减少 $v - c_i - (v - c_j) = c_j - c_i$ 钱。由于 c_i 固定，所以我们找到最大的 c_j 换过来即可。
用优先队列维护当前所有选择了优惠券的物品的 c 值，每次直接找到最大的进行比较。
时间复杂度 $O(n \log n)$，期望得分 30。

算法 3

如果你没想到寻找 2 的优化，但是你会把特殊性质 A 和特殊性质 B 并在一起，那么你可以获得一个 $O(n^2)$ 的做法。

排序后从前到后扫物品，加入新的可用的优惠券。每次在当前可以使用且未使用的优惠券中找到省钱最多的使用。注意要和折扣价比较，如果折扣价最优就选择折扣价。

再增加一个贪心。对于一个物品，我们可以把对他前面物品使用的优惠券换过来，让前面的物品改为选折扣价。暴力枚举换的优惠券，找到最优的方案和当前可以使用且未使用的比较取最优即可。

时间复杂度 $O(n^2)$，期望得分 20。

算法 4

合并算法 2 和算法 3，用优先队列维护当前所有选择了优惠券的物品的 c 值，每次直接找到最大的进行比较。
总结一下最终的贪心，对于每一个新的物品 i：

- 选择折扣价，贡献 b_i。
- 选择可以使用的未使用的最优优惠券，设其省 v 钱，贡献 $a_i - v$。
- 选择之前已经选过的优惠券换过来使用，设从物品 j 换来的，贡献 $a_i - c_i - (c_j - c_i) = a_i - c_j$。

找到最优的决策即可。
时间复杂度 $O(n \log n)$，期望得分 100。

题意简述

给你一个初始长度为 n，**只包含 $0,1,2$** 的序列 $a_{1\sim n}$，你每次可以选择两个相邻的数 p,q，删去它们，并在原位置插入 $\mathrm{popc}(p+q)$。其中，$\mathrm{popc}(x)$ 表示 x 在二进制表示下 1 的个数。显然每次操作后序列长度都会减少 1，所以执行 $n-1$ 次操作后，这个序列会恰好剩下一个数。请你最小化剩下的这个数，并给出字典序最小的操作位置序列。

此题多测，设 T 为组数，对所有数据，满足 $1\le T\le 200$，$1\le n\le 10^5$。

Author: YuJiahe

Prepared By: YuJiahe & lizhous

算法 1

我会爆搜！
时间复杂度：$\mathcal{O}(Tn! \times poly(n))$。
期望得分：12。

算法 2

我会区间 DP！设 $f_{i,j,k}$ 表示区间 $[i,j]$ 是否能得到 k $(0 \le k \le 2)$，枚举最小转移点即可。
时间复杂度：$\mathcal{O}(Tn^3)$。
期望得分：24。

算法 3

我会特殊性质 A！发现 a 序列中只有 $1,2$ 的时候 $\operatorname{popc}(p+q)=((p-1)\oplus(q-1))+1$，其中 $\oplus$ 表示异或运算。
那么无论操作顺序答案都是一样的，答案也可以直接求出，最小字典序直接操作 $n-1$ 次开头即可。
时间复杂度：$\mathcal{O}(Tn)$。
期望得分：12，结合区间 DP 后 36。

算法 4

我会判断答案！首先给出结论：

- 当且仅当 a 序列全为 0 时答案为 0；
- 序列 a 没有 0 时答案为 2 的个数对 2 取模的余数 $+1$；
- 否则，当且仅当在序列 a 为“一段 1，$2,0,2$，一段 1”（左右两边的一段 1 都可以为空）的形式时，答案为 2，其余情况为 1。

证明

前两个情况是显然的。先证明第三种情况的对应形式无法合成 1，可以发现如果 0 不参与操作只是相当于减少了一个 1，还是该形式；而 0 若参与操作，序列 a 就会变为第二种情况，并且只有奇数个 2，显然答案为 2。所以该情况一定只能合成 2。

算法 4

证明

再证明非对应形式一定可以合成。考虑归纳，如果我们能找到一个 0，然后将其左右两边合成，只要左右两边不都是 2，最终就可以合出 1。

假设 0 的个数大于两个，那么选择最左边的 0 即可，右边至少有两个 0，一定可以得到 1；

假设 0 的个数恰好有两个，那么序列 a 一定是 $A0B0C$ 的形式，其中 A, B 都是不存在 0 的序列。如果 B 为 $[2]$，那么先将 B 和旁边一个 0 合成即可得到 $A01C$ 的形式，该形式答案一定为 1；否则左右两个 0 肯定有一个合法。

算法 4

证明

假设 0 的个数只有一个。如果 0 两边至少有一个不是 2，那么我们就有操作空间了：当 2 有奇数个时，我们让最靠近 0 的 2 把经过的 1 都干掉，与 0 合并。否则我们直接让 0 与它旁边的那个 1 合并。假设这个 0 的左右两边都不合法，那么一定有一边至少有三个 2，我们把最靠近 0 的两个 2 合并，0 两边就至少有一个 1；否则可以直接合出 1。
这样所有情况都被考虑到了，证毕。

时间复杂度：$\mathcal{O}(Tn)$。
期望得分：25，结合前面算法后 52。

算法 5

考虑直接贪心，从前往后依次尝试合并，每次合并完之后重新扫一遍序列，判断答案是否变大，如果变大了就撤销这个操作。
这样时间复杂度看似是 $\mathcal{O}(Tn^3)$ 的，实际上是 $\mathcal{O}(Tn^2)$ 的。感性理解一下，第一个可以合并的位置出现的一般不会太晚，实际上第一个可以合并的位置最多不会超过 3。证明还是需要分讨，这里就不放了。
时间复杂度：$\mathcal{O}(Tn^2)$。
期望得分：36，结合前面算法后 61。

算法 6

使用平衡树一类的数据结构维护序列，每次快速找出下一个能操作的位置，或者直接暴力枚举位置。
时间复杂度：$\mathcal{O}(Tn\log n)$。
期望得分：视实现有 $48\sim92$，结合前面算法后 $70\sim94$。

算法 7

根据前面的结论，可以直接暴力维护前几个位置，后面的位置一定是连续的，这样就可以省掉 $\log$。
时间复杂度：$\mathcal{O}(Tn)$。
期望得分：100。

题意简述

有 $n+m$ 个括号序列，分别是 $S_1, S_2, S_3, \ldots, S_n$ 和 $T_1, T_2, T_3, \ldots, T_m$。

对一个括号序列 A，$f(A)$ 为满足以下条件的 (i, j) 对数：

- $i \in [1, n]$，$j \in [1, m]$；
- S_i 是 A 的**前缀**且 T_j 是 A 的**后缀**。

求所有长度为 k 的**合法括号序列** S，$f(S)$ 的和。答案对 $10^9 + 7$ 取模。

对所有数据，满足 $1 \le n, m \le 2 \times 10^5$，$1 \le k \le 10^6$，$1 \le |S_i|, |T_i| \le 5 \times 10^5$ 且 $\sum |S_i|, \sum |T_i| \le 10^7$。保证 k 为偶数。

Author: Inaba_Meguru

Prepared By: Inaba_Meguru & Daniel_lele

算法 1

我会爆搜！
时间复杂度：$\mathcal{O}(2^k \times poly(n, m))$。
期望得分：8。

算法 2

交换求和顺序，考虑我们如果钦定了一对 S_i, T_j，有多少的合法括号序列满足以 S_i 为开头，T_j 为结尾？

- 如果 $|S_i| + |T_j| \geq k$，即它们重叠，判断能否重叠且重叠后的括号序列是不是合法即可。
- 否则，我们可以在中间填入一些左右括号。这里可以直接 dp。

时间复杂度：$\mathcal{O}(nmk^2)$。
期望得分：16。

算法 3

考虑去优化那个 dp。如果我们把 (看作为 1，) 看作为 -1。那么一个括号序列合法当且仅当从 $(0,0)$ 出发，在 $(|S|,0)$ 结束且中间不越过 $y=0$。

如果钦定了前缀，后缀。相当于在图上给两个点，问不穿过 $y=0$ 每次向右上或者右下走的方案数。

这是经典的反射容斥。容易通过预处理阶乘做到 $\mathcal{O}(1)$。

对上面 S_i, T_j 重叠的情况通过哈希判断也能做到 $\mathcal{O}(1)$。

时间复杂度：$\mathcal{O}(nm)$。

期望得分：52（你能过掉特殊性质 A）。

算法 4

考虑先算重叠的情况，容易发现我们只会拿每个 S_i 的后缀和每个 T_i 的前缀进行匹配重叠，我们可以使用哈希判断是否相等。
我们还要判断 S_i 自身的左括号数量和 T_i 自身的左括号含量，还有长度就能判断是否合法，这是充要条件。
这些东西，算成三元组丢到 map 里面就行。这里复杂度是 $\mathcal{O}(L \log L)$ 的。
考虑不重叠的情况，容易发现当 S_i, S_j 的长度，和左括号的数量完全相同。我们在上面的式子中算出来的东西就是相同的。我们把这样的 S_i, S_j 划分到一个等价类里面统一计算。
容易证明，等价类个数是 $\mathcal{O}(L^{2/3})$ 的，而且远远跑不满。
时间复杂度：$\mathcal{O}(L \log L + L^{4/3})$。
期望得分：84。

算法 5

最后我们来考虑 $L = 10^7$。
首先是重叠的情况，通过使用 unordered_map 或者 gp_hash_table 之类的东西是可以做到 $\mathcal{O}(L)$ 的。但是这样子时间空间都常数巨大。
考虑常数优化，这里方法很多。我们可以先把 S_i, T_i 按照长度排序，然后我们依次枚举重叠长度，每次新枚举长度清空 map 即可。我们还可以当某一个后缀不可能使用（通过判断长度、左括号数量之类的简易判断）时就不插入 map 里面。
由于数据局限性，上面两种优化效果巨大。

算法 5

然后是最后等价类的优化。不难发现当一个等价类 (x, y) 分别表示长度和左括号数量时。我们删去 (x, y) 转而加入 $(x+1, y), (x+1, y+1)$ 效果是相同的。即我们强制钦定后面填了左括号或者右括号。

这启示我们可以根号分治，对长度 $\leq \sqrt{L}$ 的等价类通过 dp 方法推至长度为 $\sqrt{L}$ 的位置。那么这时候只会有 $\mathcal{O}(\sqrt{L})$ 个等价类。这里的复杂度就变成了 $\mathcal{O}(L)$。

由于数据局限性，把阈值往小一点开会跑的更快。

时间复杂度：$\mathcal{O}(L)$。

期望得分：100。

吐槽

本题数据不太好造。我们官方数据对大点造的都是卡满 $\mathcal{O}(L^{\frac{4}{3}})$ 的数据。可能有别的我们没有预料到的算法能够得到高分。