【题解】牛客 CSP-S 提高组赛前集训营 2

T1 服务器需求

题意

有一个长度为 n 的非负整数序列 $a_1, a_2, \ldots\ldots a_n$。每次操作可以选择最多 m个正整数，将它们的值减少 1。q次询问，每次询问会修改一个位置的值，问最少多少次操作后序列可以变为全零。询问对后面的询问有影响，操作对后面的询问没影响。

题解

解法一（$m = 1$）

注意到 $m = 1$ 时，$\sum a_i$ 即为答案。

每次询问维护差值即可。

解法二（$n, q \leq 100$）

我们可以考虑如何利用贪心策略。

注意到如果在一开始准备了很多服务器，我们总会优先选择当前使用天数最少的。

因为 a_i的值可能很大，我们可以考虑维护一个数组 b，b_j 表示当前使用过 j次的服务器还有多少台，每次从最早的开始取。

最初服务器的数量可以二分。

时间复杂度 $O(nmq\log a_i)$

解法三（$n, q \leq 1000$）

注意到其实对于每一种服务器需求，可以直接计算出服务器需要的最少数目即为

$\max(a_i, \lceil \frac{\sum a_i}{m} \rceil)$。

正确性证明

下面解释一下为什么有上面那个式子。

我们假定已知答案**x**，即需要的最少的服务器个数。显然对于任意 i，有 $x \geq a_i$ 。

接下来考虑如何安排这些服务器。对所有的服务器编号，从 1到 n。不妨将前 a_1 台服务器安排给第一天，将接下来a_2 台服务器安排给第二天，以此类推。如果第某一天安排时，安排完了所有的服务器，则再从第一台服务器开始安排。

通过这种安排方案，可以保证在 $x \geq a_i$ 的情况下不会让同一台服务器在同一天内工作两次。而让每一台服务器运行不超过 m天，就需要服务器需求的总天次（即∑ai）小于等于所有服务器可以工作的总天次（即 $x \cdot m$）。也即

$$\sum a_i \leq x \cdot m$$

考虑到 x 必须是个整数，即

$$\lceil \frac{\sum a_i}{m} \rceil \leq x$$

反推可以获得 x 的答案，取最小值即为右式

$$x \geq \mathbf{max}(a_i, \lceil \frac{\sum a_i}{m} \rceil)$$

考虑暴力修改计算，时间复杂度O(nq)。

解法四（题解做法）

我们需要一个简单的数据结构来支持删除和增加一个值并维护最大值。

利用 set 或其他简单数据结构可以维护这个操作。

时间复杂度 $O(n\log n + q\log n)$。

T2 沙漠点列

题意

给出一个 n个点，m条边的沙漠，你可以删去其中的 k 条边。求能分成的连通块数量最大值。

题解

解法一（$k = 0$）

无法删边，直接用深搜或者并查集求原图连通块数目即可。时间复杂度 $O(n)$ 或 $O(n\alpha(n))$。

解法二（原图是森林）

森林是最弱的连通图。显然每删一条边，就会增加一个连通块，因此直接输出 $n - m + k$即可。时间复杂度 $O(1)$。

解法三（题解做法）

为什么在森林上删去一条边能够多出来一个连通块呢？因为这条边删掉后，两端节点就无法连通。

换句话说，两端的节点互相连接的所有路径，一定经过这条边。也就是说这条边是割边。

能够发现，对于一条边，删除它能多出一个连通块，等价于在删除它之前的图上，它是一条割边。这两个条件可以互推。

于是我们就要在原图上先删掉所有的割边。由于保证图是沙漠，因此剩下的图一定是若干没有边相交的环。

在没有割边的时候，我们就需要去创造割边。发现在一个长度为 len 的环上删除任意一条边后，就会产生$len - 1$条割边。

那么根据很明显的贪心策略，把所有环按长度从大到小排序（使用桶排可以做到 $O(n)$ 完成排序），不断删除一个环上的边统计进答案即可。

时间复杂度 $O(n+m)$，空间复杂度$O(n+m)$。

T3 维护序列

题意

实现一个数据结构，支持以下操作:

$-1\ l\ r\ x$将 $a_l, a_{l+1}, a_{l+2}, \cdots, a\_r$ 修改为x。

$-2\ x\ y$ 查询序列中所有满足 $a_i = x, a_j = y$的点对 (i,j) 里$|\ i-j\ |$的最小值

强制在线。

题解

为了方便描述，我们将权值称为颜色。

解法一（$n, m \leq 100$）

每次查询枚举点对，复杂度 $O(n^2m)$。

解法二（$n, m \leq 1\ 000$）

一操作中，对序列暴力修改。

二操作中，从左到右对序列扫一遍。记录下目前扫到的区间 $[1,i]$中最靠右的 x,y 的位置。每次加入一个新的 x 或 y 时，查询另一个颜色最靠右的位置与它之间的距离，更新答案最小值即可。

时间复杂度 $O(nm)$。

解法三（数据随机生成）

数据随机以后，一旦经过修改，总颜色段数期望非常少。

用一个双向链表维护每一个颜色段，修改的时候将中间的颜色段删除，然后旁边两端颜色段切开（这个技巧似乎有个名字叫“珂朵莉树”？）；查询的时候类似解法二的方法从左到右暴力扫一遍。

时间复杂度比较玄学，似乎是 $O((n+m)\log n)$。

解法四（无修改操作 Ⅰ）

记颜色 x的出现次数为 cnt_x 。按照 cnt_x 根号分块。

对于出现次数小于 $\sqrt{n}$的颜色，提取出这些颜色在原序列中的位置，按照下标存到一个 vector 中排好序。每次询问时用双指针在两个 vector 里轮流推进更新最小值。由于元素个数不超过 $2\sqrt{n}$，这里复杂度 $O(n\sqrt{n})$。

对于出现次数大于$\sqrt{n}$的数，这样的颜色数量肯定不超过$\sqrt{n}$个。因此我们对这每一种颜色，都预处理一下它们到其它颜色的最近距离。也就是对每种颜色 ccc，从左到右扫，记录 c 最靠右的位置，然后每次加入一个不一样的颜色时，更新这个颜色对应的答案，这样就把到前驱的最小值统计完了。类似的操作从右到左再扫一遍就能求出到后继的最小值，进而得到预处理的答案数组了。由于种类数不超过 $\sqrt{n}$，这里复杂度 $O(n\log n)$。

因此总复杂度 $O(n\log n)$。

解法五（无修改操作 Ⅱ）

无修改操作还有一种做法。

我们将每 $\sqrt{n}$ 个元素分成一个块，总共 $\sqrt{n}$个块。然后我们把询问分成点对跨越一块和在一块内的情况分开处理。

容易发现，对于跨块的情况，我们只需要维护块内每个元素第一次出现的位置和最后一次出现的位置即可。那么配对答案时，只需从左到右扫一遍，用块内第一个元素与之前扫过的最后一个元素配对，更新最小值即可。单次复杂度 $O(\sqrt{n})$。

对于没有跨块的情况，由于块内权值只有$\sqrt{n}$ 种，那么可以预处理出两两权值之间的答案。然后询问时直接查询该块内对应答案即可。预处理 $O(n\sqrt{n})$，单次询问 $O(\sqrt{n})$。

因此时间复杂度和空间复杂度都是 $O(n\sqrt{n})$。

解法六（题解做法）

基于解法五的基础进行改进。

加上修改操作后，对于一整块的情况可以打个标记，表示整块都变成这个值了。等到询问时遇到打过标记的块，稍加特判即可。

对于两端的零散块，我们需要重建这两个块。举左侧块为例，设左侧块区间为 $[u,v]$，要修改 $[l,v]$ 之间的点为颜色 x。然后我们要更新块内的答案。注意到只有 x和 $[l,v]$中出现的颜色的答案会有改变，所以就用这些颜色去枚举块内其他颜色更新答案。这里均摊出现的颜色个数是 $O(1)$ 的，然后每种颜色都需要枚举块内其它元素，因此单次操作复杂度仍然为均摊 $O(\sqrt{n})$。

实现细节还挺多的。以下是我实现过程中遇到的一些特别需要注意的点，不然复杂度容易退化：

1. 对每个块内，我们需要对任意两种颜色都维护距离最小值。但我们无法开下 n^2 的数组，因此要将块内颜色离散到 $1 \sim \sqrt{n}$ 的大小。由于这里还有动态加入颜色和动态删除颜色的操作，因此要使用一个栈来存储还未使用的颜色标号来回收已不在块内出现的颜色标号和分配新加入的颜色标号。

2. 修改零散块时，必须要把每种颜色都恰好计算一次，如果多个位置出现了一个颜色不能重复计算。

3. 注意新标号分配与删除的时间节点，每个标号删除时需要把这个标号在块内的有关信息全部清除。

4. 由于块内的操作较为多而且冗杂，导致常数较大，因此块大小建议分小一点。实测标程的块大小在 180180180 左右才能较为稳定地通过所有测试数据。

5.(upt) 感谢 C20181210 在评论区指出，当 $x = y =$ 块修改标记时，需要特判将答案设为 0。

于是就做到了总时间复杂度 $O(n\sqrt{n})$，总空间复杂度 $O(n\sqrt{n})$解决修改和查询问题。

标程

标程可能比较难以看懂，我这里解释一下每个数组的意义。

- Ans[i][j][k]：块 i 内，颜色j与颜色 k 的最近距离。
- Lnk[i][j]：颜色j在块 i 内的标号。
- Cnt[i][j]：块 i 内颜色j的个数。用于重标号时的 0→1 的加入和 1→0 的删除。
- Left/Right[i][j]：块 i 内最左/右端颜色 j 的位置下标。

- Bel[i]：原序列中的下标 i 属于哪个块。

- L/R[i]：i 号块的左/右边界。

- Tag[i]：i 号块的区间修改标记。

- Stk[i]：用于重标号的栈。

- ColStack[i]：与 getColStack()函数配合使用，用来提取一段区间中所有出现的颜色。

- Used[i]：在 getColStack()函数中，用于判断颜色是否出现过。