基础排序算法练习题 解题报告

杭州学军中学 金策

1 试题来源

2016年集训队互测

2 试题大意

对于一个长度为n的数组$a[1], a[2], \cdots, a[n]$，定义这样一个算法：它包含m个步骤，其中第i步是将$a[l_i], a[l_i+1], \cdots, a[r_i]$原地升序排序。

共有q个询问，每个询问中给出一个初始数组a，你需要判断上述算法能否将这个数组升序排序（即最终是否满足$a[i] \le a[i+1], i = 1, 2, \cdots, n-1$）。

3 数据规模

编号	$n \le$	$m \le$	$q \le$	备注
1	100	100	100	
2	200	200	200	
3	1400	900000	8	
4	1500	1000000	8	
5	1400	4000	1400	
6	1500	5000	1500	
7	1200	700000	1200	
8	1300	800000	1300	$a[1..n]$ 由$n/2$个1和$n/2$个0组成，n是偶数
9	1400	900000	1400	
10	1500	1000000	1500	

对于所有数据，$1 \le n \le 1500, 1 \le m \le 1000000, 1 \le q \le 1500, 1 \le l_i \le r_i \le n, 0 \le a[i] \le 1500$。

4 算法介绍

4.1 算法一

首先考虑最朴素的算法。对于每次询问，暴力执行算法的m个步骤，每个步骤用$O(n^2)$的冒泡排序法或插入排序法实现。最后检查一遍数组是否有序。

时间复杂度是$O(qmn^2)$，可以通过20%的数据。

4.2 算法二

考虑对算法一进行优化，将排序算法改成$O(n \log n)$的快速排序。

时间复杂度是$O(qmn \log n)$，可以通过20%的数据。

4.3 算法三

在测试点$3, 4$中，q的规模较小，而n, m的规模较大，所以我们需要优化询问一次的复杂度。

回顾一下插入排序法的算法流程，每次对于相邻两个元素$a[i], a[i+1]$，如果$a[i] > a[i+1]$，则将它们交换。注意到每次这样的交换都会使a数组的逆序对数目减少1，而a数组的初始逆序对数量是$O(n^2)$级别的，因此在整个算法过程中我们至多进行了$O(n^2)$次交换。

但这样还不够，因为我们每轮排序时要对$a[l_i..r_i]$进行一次扫描，需要$r_i - l_i = O(n)$的时间。如果这段序列中逆序对数较少，那么这趟扫描的时间是浪费的。

因此，为了高效维护，可以把当前所有$a[i] > a[i+1]$的i都插入到一个平衡树（或STL的set）里。每次将$a[l..r]$进行排序时，找出set中不小于l且最小的i，如果$i \ge r$，说明$a[l..r]$已经有序，可以结束操作；否则将$a[i], a[i+1]$交换，然后检查$a[i-1], a[i]$和$a[i+1], a[i+2]$的大小关系，并在set中进行更新。

于是总的时间复杂度是$O(q(m+n^2)\log n)$，可以通过40%的数据。

另外，将上面的set换成vEB树可以得到更优的理论复杂度。

4.4 发现性质

剩下的测试点中q的规模都较大，对每个询问进行模拟看上去比较困难。我们可以换一个思路，转而研究具有什么性质的初始序列是可以被排序的。

注意到在8号测试点中我们排序的是一个01串，可以先从这种情况入手考虑。设01串中1的数量为k。

如果第m次操作后，数组被排好序。那么在第m次操作前，数组必然是前面一段为0，后面一段为1，而中间$a[l_m], a[l_m+1], \cdots, a[r_m]$这一段可以任意打乱顺序。同理，如果确定了第$m-1$次操作结束后的结果，那么此时$a[l_{m-1}], \cdots, a[r_{m-1}]$这一段数应该是有序的，我们可以把它们任意打乱顺序，得到第$m-1$次操作前的所有可能结果。如果按这样从后往前递推，一直推到第1次操作之前，理论上就可以得到所有可被排序的初始序列。但问题是“任意打乱顺序”可以得到的序列有很多种，要把它们全部显式地进行遍历是不现实的。因此，可以尝试从中寻找一个特定的序列作为“代表”元素，用它来表示出所有可被排序的序列。也就是说，它是所有可行序列中情况最坏的；如果某个序列比它好，那么这个序列也能被排序。于是首先需要给出“好坏”的定义。

我们考虑的全集U是所有包含k个1的n位01串。对于01串$a \in U$，用$\mathrm{pos}(a, i)$表示a中从左到右第i个1所在的位置。对于两个01串$a, b \in U$，定义a比b优秀，当且仅当对于$i = 1, 2, \cdots, k$都有$\mathrm{pos}(a, i) \ge \mathrm{pos}(b, i)$。这是一个偏序关系。直观来说，就是可以通过将$b$串的某些1右移而得到$a$串。例如，100111就比101101优秀。

对于01串$a \in U$，定义a生成的集合为所有比a优秀的01串组成的集合。

观察刚才的递推过程，可以感受到，打乱顺序的最坏方式就是将$a[l_i], \cdots, a[r_i]$的1全部挤到左边，0放到右边（即按降序排序），任意一种打乱方式都比这种更加优秀。接下来我们更加详细地说明这一点。

先来定义一些记号。用$f_{01}(a, l, r)$表示将01串a的第l项到第r项升序排序后得到的串，用$f_{10}(a, l, r)$表示将a的第l项到第r项降序排序后得到的串，$l \le r$。容易发现这样一些性质：

(1) $f_{01}(a, l, r)$比a优秀，a比$f_{10}(a, l, r)$优秀。

(2) 令$b = f_{01}(a, l, r)$，则$f_{10}(a, l, r) = f_{10}(b, l, r)$。

(3) 对于a, b串和l, r，如果a比b优秀，那么$f_{01}(a, l, r)$比$f_{01}(b, l, r)$优秀，$f_{10}(a, l, r)$也比$f_{10}(b, l, r)$优秀。

用A_k表示所有能被第$k+1,k+2,\cdots,m$次操作排好序的01串集合（即第k次操作结束后所有可能的结果）。我们的目标就是找出集合A_0的性质。

显然A_m中只包含一个串，即$n-k$个0后面跟着k个1，把这个串记为a_m。再定义一系列串$a_{m-1},a_{m-2},\cdots,a_0$，其中$a_{i-1}=f_{10}(a_i,l_i,r_i)$。根据我们刚才的直观感受，容易发现$a_i$生成的集合即为集合$A_i$。这可以用归纳法证明：

如果串$x\in A_i$，根据定义有$y=f_{01}(x,l_{i+1},r_{i+1})\in A_{i+1}$，即$y$比$a_{i+1}$优秀，再根据刚才的性质(3)有$z=f_{10}(y,l_{i+1},r_{i+1})=f_{10}(x,l_{i+1},r_{i+1})$比$f_{10}(a_{i+1},l_{i+1},r_{i+1})=a_i$优秀，又由于$x$比$z$优秀，因此$x$比$a_i$优秀。

另一方面，如果串u比a_i优秀，根据性质(3)可知它第$i+1$次操作后的结果$v=f_{01}(u,l_{i+1},r_{i+1})$比$w=f_{01}(a_i,l_{i+1},r_{i+1})$优秀，又由$a_i=f_{10}(a_{i+1},l_{i+1},r_{i+1})$，易知$w$比$a_{i+1}$优秀，于是$v$比$a_{i+1}$优秀，即$v\in A_{i+1}$。因此$u$可被第$i+1,\cdots,m$次操作排好序，即$u\in A_i$。

于是我们就证明了a_i生成的集合即为集合A_i。

4.5 算法四

根据上面发现的结论，我们得到了针对8号测试点的特殊算法：预处理出串a_0，然后每次判断输入的01串是否比a_0更优秀即可。

其中第二步可以容易地$O(n)$实现；第一步预处理中，我们需要高效实现给一段区间中的01排序，这可以使用算法三实现。

时间复杂度是$O((m+n^2)\log n+qn)$。

4.6 算法五

不妨假设待排序的数组a是一个$1\sim n$的排列。如果不是的话，可以先将它离散化，其中值相同的元素按照下标递增的顺序分配。容易看出这样并不会影响最终的答案。

现在尝试将算法四进行推广。对于初始排列$a[1..n]$和给定的k，可以定义一个01串b_k，其中$b_k[i]=1$当且仅当$a[i]\geq k$。可以看出，算法能将a数组正确排序，当且仅当对于所有$k=2,3,\cdots,n$，该算法都能将01串b_k正确排序。

于是，根据算法四，现在需要对于每个k，预处理出$n-k$个0后跟k个1的串倒着进行m次操作后的结果c_k。直接对每个分别预处理肯定太慢了，但我们可以做一件等价的事情：令初始排列为$1,2,\cdots,n$，按i从m到1的顺序，每次将第l_i项

到r_i项降序排序。最后得到的序列中，将小于k的值用0代替，其他值用1代替，得到的01串就是所求的c_k。这样做的正确性是显然的，而且可以由此得知c_{k-1}是由c_k把某个位置的0改为1而得到的。

现在要对于所有$k = 2, 3, \cdots, n$检查b_k是否比c_k优秀。这是一个经典问题，可以将b_k中的1取负号，c_k中的1取正号，每次加入一对新的$+1, -1$后，检查数组的前缀和是否非负。这可以用经典的线段树区间加、维护区间最小值来实现。时间复杂度是$O(n \log n)$。

预处理的过程可以用算法三实现。因此算法的总时间复杂度为$O((n^2 + m + qn) \log n)$，可以通过100%的数据。

4.7 其他部分分算法

如果选手发现了上面的结论，但不会使用算法三中的消逆序对技巧进行预处理，也可以拿到比较可观的分数。例如在第5、6号测试点中，m的值较小，可以直接每次暴力排序；在8号测试点中，由于待排序的数字都是0和1，所以也可以转化成区间求和与区间赋值操作，用线段树实现。

另外也有一种可能通过5、6号点的做法：考虑用数据结构维护这个序列，将它分成若干段，使得每段都是单调递增的，并把一段中的数字放进一棵平衡树维护。进行区间排序操作时，只要在区间边界处断开，然后将中间的若干段进行归并，归并时使用一些合并平衡树的技巧（参考2015年王逸松的互测题解）。但是这个算法的复杂度并不容易分析。

还有一种做法是将永远用不到的操作给去掉，然后暴力模拟每个询问。但是大部分数据是经过构造的，没有用的操作数量并不多。

利用这些部分分算法，结合一定的优化，预计可以得到$50 \sim 70$分，具体得分多少取决于优化的方式以及选手的能力与经验。

5 总结

这道题看上去与以往的许多数据结构题类似，需要维护一个序列并在上面支持区间操作；但是又和某些数据结构裸题不同，这题的瓶颈并不在于高深的数据结构理论或复杂度分析技巧，而是在于研究题中操作的性质，倒过来进行分析，从而发现一些结论。然后再根据这些结论的需要，选用经典的数据结构

进行维护。本题标算中用到的数据结构有set以及最基本的懒标记线段树，代码难度并不高，大多数NOI水平选手应该都能熟练使用。但本题有一定的思维难度。

本题的分数估计：$1 \sim 3$名选手获得$80 \sim 100$分的高分，三分之二左右的选手能获得至少40分，所有选手都能获得至少20分。

整场比赛的分数估计：估计最高分在160分左右，中位分在100分左右。