NOIP 模拟赛 题解

by Comentropy

2025 年 9 月 25 日

题目名称	微光	泪雨	深海	虚无
目录	`glow`	`namidame`	`sort`	`nihility`
原题	P8048	P12977	无	P13866

提交源程序文件名

对于 C++ 语言	`glow.cpp`	`namidame.cpp`	`sort.cpp`	`nihility.cpp`

编译选项

对于 C++ 语言	`-O2 -std=c++14 -static`

微光（glow）

【算法一】

容易想到一个经典的手段：我们不处理反弹，而是把两个相碰的泡泡看作是互相穿过。于是在两个泡泡相碰之后，先前（在左侧）向右运动的泡泡颜色可以视为不变，而先前向左运动的泡泡颜色变为 $(a+b) \bmod k$。

暴力处理向左向右泡泡的碰撞，即可通过前四个测试点。

【算法二】

每个向左的泡泡总是从右到左依次与向右的泡泡碰撞，颜色数 k 很小，且在经过同一个向右泡泡时，相同颜色的泡泡行为相同，所以我们尝试统一处理。

扫描线，从右往左处理泡泡，容易处理每种颜色的泡泡的位置之和、个数，于是计算答案是简单的，实现时注意处理边界。从左向右维护映射也是可以的。

泪雨（namidame）

【算法一】

枚举回文中心，随后向两边扩展，可以通过前四档部分分，期望获得 40 分。

值得注意的是，如果已知回文中心，我们显然利用对称性，可以只算问号个数。

【算法二】

小常数 $O(n\log n)$，实现优秀可以直接通过。

我们把区间 $[l,r]$ 看成一个点，那么我们遇到的有两个限制，以下假设枚举的是回文中心 l：

1. 问号个数 $\geq$ 确定字符个数，设前者的前缀和 A，后者前缀和 B，则 $A_r - A_l \geq B_r - B_l$，那么有 $A_r - B_r \geq A_l - B_l$。
2. $r - l + 1 \leq$ 回文半径。

于是转化成了二维数点问题。按 $A_i - B_i$ 从大到小的顺序加入点（相同的要先加完再统计）。对于某个回文中心 l，假设其右端回文最远能到 r，那么，就统计 $[l,r]$ 内可能的右端点有多少个，以及对应的问号的前缀和的和，这样利用对称性就容易计算。需要注意奇偶长度的区别。时间复杂度 $O(n\log n)$。

【算法三】

注意 Manacher 实际上是能够表示出所有的回文串的。

在其转移过程中，当前这一段如果没有被右边界截断，那么显然可以直接转移（如果我们知道了左边对称过来的串的权值）。

那么如果被右端截断了呢？你可以直接使用 `hash_table`，在发生扩展时，把所有信息都计算并存下来（记录 `hash` 值，和问号个数），截断的时候再提取出来（该状态必然出现过）。由于扩展次数是 $O(n)$ 的，所以这个做法时间是期望 $O(n)$ 的。

考虑我们其实不需要存这么多状态，对于一条转移形成的链，下去的时候扩展，上来的时候回溯，时间是线性的。所以把 Manacher 所有的转移建出操作树，然后在树上 dfs（这样可以直接求出回文半径为某个长度的信息），每次向下搜索将扩展了长度的回文串的对应答案更新，回来的时候回溯就行，时间复杂度和扩展次数同级别，仍然是 $O(n)$ 的。

然而，这样做需要你会一些快速的遍历方式。**我们有一种常数更优秀的做法。**

在截断的时候进行暴力收缩——这样的复杂度其实是正确的。记回文半径为 d_i，从 $p_i = 2\times mid - i$ 处转移，我们考虑当前最右的区间 $[l,r]$，我们在 $i + d_i \geq r$ 时就更新回文区间和对应中心。当 $i + d_{p_i} > r$ 时发生收缩，总的收缩长度：

$$\sum i + d_{p_i} - r \leq \sum i + (r - p_i) - r = \sum i - p_i = \sum 2(mid - i)$$

如果发生了收缩，mid 就会变成 i，所以 $\sum 2(mid - i) \leq 2n$，复杂度是对的。

这个做法在某些情境下貌似比我先前用操作树的做法强，它可以完成之前联考搬的题：逆にする関数。（注：我出泪雨的时候还没做过这道题）。

深海（sort）

【前言】

不知道有没有公开来源，但是有类似的题目推荐一做：[pjudge NOIP Round #7] 冒泡排序。

【$o = 2$】

研究排序问题时（尤其是排列），可以先进行值域压缩：通常我们会把序列变成 `0-1` 序列，比较特殊的时候还会有压缩成 $[-1, 1]$ 或其它更大一点值域的需求，在此不展开。我们二分值域，随后把序列变成 $a'_i = [a_i \geq mid]$，接下来只需要判定询问的位置是 1 是 0。

考虑刻画一下冒泡排序的操作。对于下标从 1 到 n 的序列：

现在 1 之间没有区别，我们完全可以让最左边的那个 1 不断交换到最后，也相当于把最左边的 1 之后的下标减 1 再把它的下标设成 n。注意一些边界，那么判定就是简单的了：

- 如果 $k \geq$ 序列中 1 的个数，那么显然序列已经排序完了。（不用特别判这个）
- 如果 $x + k > n$，那就说明此时 x 在长度为 k 的后缀上，只需要判断 1 的数量是否足够。
- 其余情况，我们只用数一下 $x + k$ 前面有多少个 1，若只有 $< k$ 个 1，那答案显然就是 0；否则，答案就是经过了 k 轮平移后到达 x 的位置：a'_{x+k}。

你很快就会发现二分是不必要的：如果我们二分到的答案不大于 $x + k$ 前的第 k 大的数，那么它对应的答案就是 a_{x+k}；否则，判定答案必然返回 0。所以你只需要支持区间第 k 大查询即可。

对于区间询问，使用主席树即可做到 $O((n + q) \log n)$。

【$o = 1$】

由于涉及了具体的位置，我们不妨把值域压缩成 $[-1, 1]$ 再观察（其实可以脱离值域压缩观察，因为只涉及和 x 比大小）。我们考虑 0 的动向。

首先，当前面有 1 时，和 $o = 2$ 一样，它的行为和 -1 一致，每次会向前移动一步；当前面没有 1 的时候，它向后冒泡，直到遇见一个 1，之后这个 1 就会向后冒泡。

同样地，考虑形式化这个东西：

- 如果 $k \geq$ 序列中 $0, 1$ 的个数，那么显然序列已经排序完了。
- 先统计现在 x 的位置前有多少个 1，记为 c，$c \geq k$ 的情况是平凡的，现在还要向后移动 $c - k$ 次，我们要找到从左到右的第 k 个 1 的位置 p，计算答案直接做情况比较复杂，换个方向考虑——考虑后面有多少个 -1 会移到它前面，显然就是从 x 的位置到 p 中的 -1，这样就做完了。

这一部分可以离线后扫描线，主席树也可。

时间复杂度 $O((n + q) \log n)$。

虚无（nihility）

【观察】

这个牌堆就是一个栈的形式，对于暴力来说我们可以把栈的状态存下来。这样状态实在太多。容易发现，如果在 $n-1$ 处加入所有颜色的自环，我们就只需要判定能否空栈到达点 $n-1$。

同样地，我们希望做一些操作使得图的有效状态数变少。

【算法一】

如果可达，那么答案必然 $\leq n-1$，考虑在起点前加一些“没有限制”的虚点，以此使得可以在起点 0 获得所有可能方案，我们这样连边：

- 对于 n 号点，向 0 连所有颜色的边。
- 对于 $n+i(1 \leq i \leq n-1)$ 向 $n+i-1$ 连所有颜色的边。

没有限制的意思就是：这些边只负责加牌，而不执行操作 1。这样如果我们能够从虚点 u以空栈状态到达 $n-1$，答案就是 $u-(n-1)$（若从 0 能到答案就是 0）。

现在问题转成了一个可达性问题，我们只需要判定能否空栈地从起点，以空栈的状态到达终点。注意空栈任选走边加牌到非空栈强制走边这个过程比较特殊，我们需要在状态里加入这个信息。

记 $f_{i,j,c}$ 表示，存在一条路径使得空栈状态的 i 可以到达空栈状态 j， 且，路径中不存在一个空栈的时刻有颜色为 c 的出边。转移考虑类似括号序列生成的转移。

第一种转移形似传递闭包，也可看作把两个括号序列拼在一起：$f_{i,j,c} \leftarrow f_{i,k,c} \wedge f_{k,j,c}$。

第二种转移形似在括号序列两侧加一对括号：如果 i 有一条颜色为 c 的入边 (u,i)，j 有一条颜色为 c 的出边 (j,v)，且 u 不存在一条颜色为 c' 的出边，$f_{u,v,c'} \leftarrow f(i,j,c)$。注意此类转移对我们建的虚点也生效，因为它们没有要强制走之类的限制，也就是 $u > n-1$ 时也执行转移。

注意一种情况：当可能用满所有颜色边的时候（其实就是 0 连出了所有颜色的边的时候），我们新建一个颜色 k，即可解决这种情况。

转移顺序比较复杂，使用记忆化搜索即可。时间复杂度 $O((n^2+nk+mk)nk)$。