IOI 2012 解题报告

南京师范大学附属中学　顾昱洲

目录

1 odometer

1.1 题目大意

有一$256 * 256$的网格，初始时有一机器在左上角，且面朝北方。每个格子中有$0 \sim 15$个石子。你需要编写一些程序解决一些任务。

程序编写需要满足一定规则。

程序由多行组成。每行“#”之后的部分为注释。剩余部分为语句，或空行。

六条基础语句：

left 左转

right 右转

move 前进一格(若存在)

get 若当前格内有石子，那么扔掉当前格内的一个石子

put 向当前格内加入一个石子(石子的来源是虚无)

halt 退出程序

结构控制语句：

L: 定义一个标签L

pebble L 若当前格内有石子，跳转到标签L处

border L 若前方为边界，跳转到标签L处

jump L 无条件跳转至标签L处

合法的标签L为长度不超过128的由$a, \ldots, z, A, \ldots, Z, 0, \ldots, 9$组成的字符串。

(如果不将标签定义视为一条语句，将较容易理解。)

程序从第一行开始运行，如果没有跳转语句，那么在一条语句运行结束后运行下一行。若到达程序尾部，那么结束程序。

评分依据主要有两点：

程序长度(PS) 即去除注释后非空行以及非标签定义行的行数

运行长度(EL) 即程序运行至结束总共进行的非定义标签的操作的次数

以下为五个子任务：

子任务一 在$(0,0)$以及$(0,1)$各有X个和Y个石子，其余格子都没有石子。若$X \leq Y$，程序结束时停留在$(0,0)$，否则应停留在$(0,1)$。$PS \leq 100, EL \leq 1000$

子任务二 同子任务一，但是结束时$(0,0)$和$(0,1)$应各有X个和Y个石子。$PS \leq 200, EL \leq 2000$

子任务三 初始时在$(0,x)$和$(0,y)$ $(x \neq y$且$x+y$为偶数)各有一个石子。结束时应停留在$(0, \frac{x+y}{2})$。$PS \leq 100, EL \leq 200000$

子任务四 整个地图中总共有不超过15个石子。将所有石子移至$(0,0)$。有部分分。满分要求$PS \leq 200, EL \leq 200000$

子任务五 找出地图中石子数最少的点，并停留在那里(若有多个选择任意一个)。并且结束时地图应与初始时相同。有部分分。满分要求$PS \leq 444, EL \leq 44400000$

1.2 算法分析

子任务一 每次先判断$(0,0)$是否为空，若是则退出，否则删除一颗石子；然后判断$(0,1)$是否为空，若是则退出，否则删除一颗石子。

子任务二 同子任务一，但是在删除一颗石子时，在该格下面一格加入一颗石子。并且在退出前将这些石子加回去。(要对两种不同结果写两个退出子程序)

子任务三 每次，先找到左边的有石子的格子，然后将这个石子向右移一格；然后找到右边的有石子的格子，将这个式子向左移一格。若某格有两个石子，那么就停止在这格。

子任务四 首先我们得到一个运行结果正确的程序。一个显然的想法是先将所有行的石子都移到该行的第一列，然后再将第一列的所有石子都移到左上角。

本题对EL的要求非常严格。地图中总共有$256 * 256 = 65536$个格子，在每个格子上进行的操作至多有$\frac{200000}{65536} = 3.05$个。由于还需要移动石子，因此实际上每个格子只能进行两个多一点的操作。

每个格子必须要进行pebble和move，这样就已经占用两个操作了。border看似是每个格子必须的操作，但是实际上，由于地图大小是固定的(256*256)，我们可以每8个格子才判断一次。(当然，这样的话，8个格子的move和pebble以及相应的操作都需要写出来。8个格子判断一次是综合了PS限制的考虑的一个合理的值。)

由于EL的严格限制，遍历时每个格子只能访问一次。也就是说，我们只能两行两行处理，将这两行的所有石子都移到下一行的第一列。

之后仔细实现即可。

子任务五 EL的限制非常宽，基本相当于没有。我们先来考虑如何得到一个运行结果正确的程序。

根据子任务一，我们可以有一个初步的想法：做多轮，每轮遍历整个地图，若某位置为空则停在某位置，否则在该位置删除一颗石子。

这里需要将地图还原。在子任务二中，我们使用了一个计数器。但是在这里，地图的所有位置都有可能是答案位置，找不到空的位置给我们当做计数器。

在程序相同的位置，是没有区别的，我们无法区别各轮。因此，一个自然的想法是在程序中记录轮数。程序中可以使用的记录工具，只有行数。因此，对于每一轮，我们要分别写一段程序。换一句话说，我们是枚举最小值，然后判断地图中是否存在某个位置的值等于这个最小值。

实现时的一些技巧：

- 实现较短的遍历。一行一行处理，每个格子访问两遍(当然，只处理一遍)。
- 这样实现的遍历，一次遍历结束时，程序位置在最下面一行的第一列。我们这时候可以不用再回到第一行，而是直接转向。对于程序而言具体的坐标是没有意义的。
- 只需要对最小值为0到14进行运算。最小值为15时可以不进行任何处理。

• 判断每个格子值是否为x时，我们先get x次，再pebble。无论是否为空，我们都需要再put x次。这样看上去每次判断要消耗$x*3$个行。但是实际上，pebble为空的时候，我们可以优化掉一些put。例如我们在$x=4$时在某格pebble为空。我们先put，然后jump end_3 ，end_3 的操作是put以及jump end_2, ..., end_0 的操作是halt。这样总共只需要约15*2=30行。

2 rings

2.1 题目大意

有$n(\leq 10^6)$个点。初始时任意两点间没有边。现在有两种操作：

(1) 在某两点间加一条无向边(无重边，无自环)；

(2) 询问，存在多少个点，使得删除这个点以及与这个点相连的边后，该图为一些链(孤点也算链)。

操作总数$\leq 10^6$。必须在线。

2.2 算法分析

如果我们已知必须要删除某个点，那么很容易维护当前的图是否为一些链(通过记录并查集以及每个点的度)。

如果在某次操作以后，某个点的度变为3。那么以后可能为答案的点，只有该点以及与该点相邻的点(4个)。对这四个点，我们分别维护删除该点后的合法性。复杂度为每次操作$O(\log n)$ (4个并查集)。

若没有度大于2的点，那么图为一些环和链。此时有三种情况：

(1) 图无环。此时答案即为n。

(2) 图恰好有一个环。此时答案为该环上的点数。

(3) 图有多个环。此时答案为0。

这是容易维护的。

时间复杂度$O(n\log n)$，空间复杂度$O(n)$。

3 scrivener

3.1 题目大意

有一台打字机，维护一个字符序列(初始为空)，有两种操作：

(1) 在当前的序列末尾加入一个字符；

(2) 撤销之前的U_i个操作。

撤销操作本身也可以被撤销。

此外，在某些操作之后，你要回答一些询问，询问内容是当前序列的第l_i个字符。

必须在线。

总操作个数$n \leq 10^6$。

3.2 算法分析

考虑所有操作组成一棵继承树。维护每个操作后的终止结点，以及序列长度。

操作(1)即为在当前终止结点下增加一个孩子，并令新的终止结点为这个孩子，新的序列长度为当前序列长度+1。操作(2)即为将新的终止结点赋值为$U+1$个操作之前的终止结点，序列长度同理。

接下来处理查询操作。我们记录每个结点的第2^k个祖先。查询即为当前终止结点的某个距离已知的祖先对应的字符。

时间复杂度$O(n\log n)$，空间复杂度$O(n\log n)$。

4 city

4.1 题目大意

无限网格地图上有$n(\leq 10^5)$个黑色格子，其余为白色格子。满足所有黑色格子四连通且所有白色格子四连通。

定义两个黑色格子x与y的距离$d(x,y)$为最小的l使得存在黑色格子序列$a_0,a_1,\ldots,a_l$ 满足$a_0=x,a_l=y$，且对于任意$0\leq i<l$满足a_i与a_{i+1}四连通。

令所有黑色格子为$A_0,\ldots,A_{n-1}$，求

$$\sum_{0\leq i<j<n} d(A_i,A_j)$$

答案模10^9。

4.2 算法分析

DP。

考虑固定一个起点格子x，对于其他的每个格子y，有八种情况：

- x到y的所有最短路第一步均向上(下、左、右)；
- x到y的所有最短路第一步均向上或向右(右下、下左、左上)。

对于这八种情况，我们分别记录满足这种情况的格子个数，以及到x的总距离。

以向上和上右两种情况为例描述转移方式：

我们记x上下左右的的格子分别为$UP(x)$、$DOWN(x)$、$LEFT(x)$、$RIGHT(x)$。

仅当$UP(x)$存在时，存在从x最短路第一步必须向上的格子，有：

- 从$UP(x)$最短路第一步必须向上的格子；
- 若$LEFT(x)$为白色格子，则从$UP(x)$最短路第一步必须向左的格子、以及最短路第一步可以向上或向左的格子；
- 若$RIGHT(x)$为白色格子，则从$UP(x)$最短路第一步必须向右的格子、以及最短路第一步可以向上或向右的格子。

仅当$UP(x)$、$RIGHT(x)$、$UP(RIGHT(x))$均存在时，存在从x最短路第一步可以向上或向右的格子，为$UP(RIGHT(x))$最短路第一步必须向上、必须向右、或可以向上或向右的格子。

其他情况均与以上某种情况对称。正确性显然。转移的具体实现显然。

实现时可以对每个下标，预处理出其上下左右的格子的下标。以后可以不管具体坐标，只用下标进行运算。

不管这步预处理，时间复杂度与空间复杂度均为$O(n)$。

5 supper

5.1 题目大意

有一个大小为K的调色板，总共有N种颜色。初始时调色板上有0到$K-1$的颜色。现在要向调色板上加入N次颜色。每次加入时：

(1) 若该颜色调色板上已有，则忽略这次操作；

(2) 否则删去调色板上的一种颜色，并加入这种新颜色。

你需要写两个程序。第一个程序可以知道K、N、以及N次加入的颜色分别是什么。(即第一个程序可以通过离线算法得到最优方案。)

然后第一个程序向第二个程序传输一个长度不超过M的二进制串，第二个程序只知道K和N以及被传进来的二进制串。之后每次读入一个颜色，并返回当前的操作。(即第二个程序必须在线处理。)

第二个程序所进行的(2)操作数量不能比最优方案多。

满分算法需要通过$M=2N$的数据。$N \leq 10^5$。

5.2 算法分析

首先我们得出一个最优方案。原题中给出了一个最优方案：每次加入时，若为情况(2)，则删除调色板上在加入序列中下一次出现最晚的颜色。我们在这里不给出这个方案最优性的证明，而是直接使用它。

得到一个最优方案后，我们根据这个最优方案，对于初始的K个位置(也就是颜色0到$K-1$)以及N次加入，记录这个初始位置(或加入)在之后是否被一次(1)操作使用过。第二个程序运行过程中维护K个位置在之后是否被(1)操作使用过。

“使用过”的含义是：如果之后某次加入的颜色与这个初始位置(或加入)相同而忽略，那么我们认为这个初始位置(或加入)被使用过。

每次加入时，先判断调色板上有没有这种颜色，如果有的话将是否使用过更新为这次加入的值。如果没有，找到调色板上任意一个之后没有被(1)操作使用过的位置，然后删去这个位置。

证明 正确性：我们的方案的每一个时刻，之后被(1)操作使用过的维护的颜色组成的集合与最优方案该时刻的集合相同；以及没有被(1)操作使用过的位置数与最优方案相同。

因此在最优方案出现情况(1)操作时，我们的方案也是(1)操作；最优方案出现(2)操作时，我们的方案也是(2)操作，且必定可行。

最优性：该方法得到的解与我们的最优方案的(2)操作总数一致。

证毕。

传输长度$N+K$，时间复杂度$O(n\log n)$，空间复杂度$O(n)$。

6 tournament

6.1 题目大意

有一个竞赛，$n(\leq 100000)$名选手参加。每名选手有一个两两不同的$0\ldots n-1$的能力值。初始时他们按照某种顺序(可能是乱序)站位。竞赛进行C场，每场选择仍然存活的第L_i到R_i个选手进行竞赛，能力值最高的选手获胜，其他选手则落败并退出比赛。保证C场结束后只剩下1名选手。

现在有一名能力值为R的选手迟到。其他选手都已经排好了位置。你要将这名选手插进某个位置，使得这名选手获胜的场数最多。若有多个获胜场数相同的位置，选择最左的位置。

6.2 算法分析

首先，将每场的L_i和R_i转换为初始序列的下标。即，对于每个L_i和R_i，求出L_i'和R_i'，使得第i场比赛的胜者为原序列中L_i'到R_i'之间的最强者。这可以通过维护一棵线段树得到。

然后，我们考虑维护对于每个位置，我们将迟到选手插入到这个位置后，他能够获得的胜利场数。显然，每一次L_i'到R_i'之间的比赛，只会影响插入位置在L_i'到R_i'之间的胜利场数。此时，原序列(长度为$n-1$)中与迟到选手比赛的选手为L_i'到$R_i'-1$之间的选手。

若该区间内有能力值大于R的选手，那么L_i'到R_i'之间的插入位置被标记为退出比赛；否则，将尚未退出比赛的L_i'到R_i'之间的插入位置的获胜场数加一。

这是容易用线段树维护的。

时间复杂度$O(n\log n)$，空间复杂度$O(n)$。