《会议》解题报告

时庆钰

October 2022

1 简要题意

在线维护一个可重集合 S，要求支持以下操作：

- 给出 x，你需要在可重集合中插入一个元素 x。
- 给出 i，你需要在可重集合中删除在第 i 次操作中插入的元素。保证第 i 次插入操作在本次操作时存在，且在该操作前没有被删除。

记 $c_S(x)$ 表示可重集合 S 中元素 x 的出现次数。我们称 $x \in S$ 为可重集 S 的绝对众数，当且仅当 $2 \cdot c_S(x) > |S|$。

在每次操作后，你需要：

- 报告可重集合 S 是否存在绝对众数。
- 给出集合 S 的绝对众数。

在每次操作后，你可以向交互库进行以下询问：

1. 给出元素 x, y，交互库返回是否有 $x = y$。
2. 给出元素 x, y，交互库返回是否有 $x \leq y$。

记在每次操作后与回答询问前，所进行的 1 操作（相等判定操作）的次数的最大值为 E，2 操作（比较操作）的次数的最大值为 C，则最终的得分与参数 E 与 C 的值有关。

2 算法介绍

2.1 算法 1

考虑朴素算法。我们在每次插入一个新的元素时，通过相等判定操作得到其与所有其他元素的相等关系。这样，我们便可以实时维护出绝对众数的存在性以及绝对众数的值。

该算法的时间复杂度为 $\Theta(q^2)$，所需要的总询问次数也为 $\Theta(q^2)$。期望通过子任务 1。

操作类型	E-代价	C-代价	时间复杂度
插入	$\Theta(\|S\|)$	0	$\Theta(\|S\|)$
删除	0		
合计	$\Theta(q^2)$	0	$\Theta(q^2)$

表 1: 算法 1

2.2 算法 2

为了方便表述，我们使用以下图例来描述一个状态：

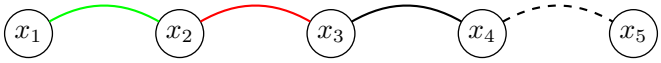

图 1: 描述结构的图例

- 图中的每个顶点 x_i 代表一个元素。
- 图中每条绿边（如边 (x_1, x_2)）代表两端点的元素之间的关系已知，且他们的关系为相等。
- 图中每条红边（如边 (x_2, x_3)）代表两端点的元素之间的关系已知，且他们的关系为不等。
- 图中每条黑边（如边 (x_3, x_4)）代表两端点的元素之间的关系已知，且我们不关心他们是否相等。
- 图中每条用虚线表示的边（如边 (x_4, x_5)）代表我们需要询问这两条边的关系。

我们考虑将所有相等的元素分成若干组，并将每个组之间按照大小关系从小到大组成一条链。这样，我们可以轻易得知每类元素的出现次数。下图展示了一个可能的局面，其中所有由绿色边构成的联通块均代表了一类相同的元素。

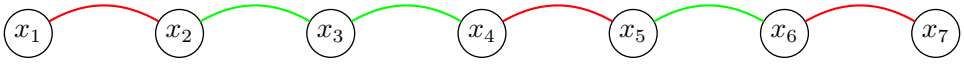

图 2: 一个可能的局面

对于插入操作，我们在链上二分出最后一个小于等于 y 的元素的位置，并通过一次相等判定操作来更新极长的段即可。而对于删除操作，我们只需要定位到对应的位置，并将相应的元素删除即可。

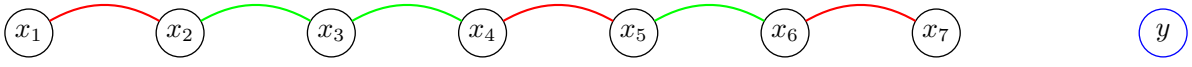

图 3: 在当前的结构中插入元素 y

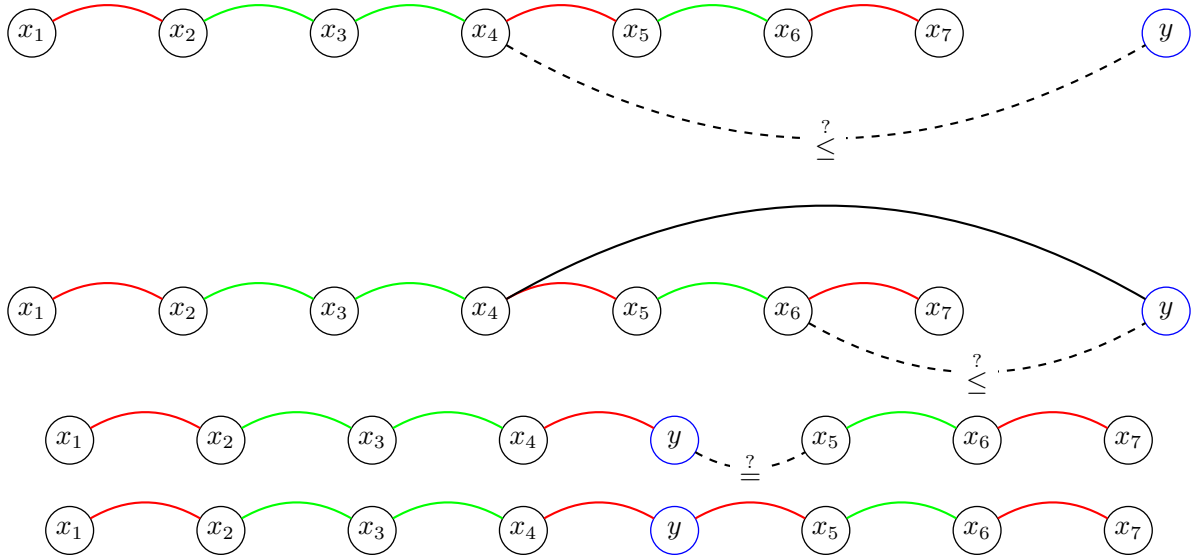

图 4: 插入 y 的一组可能的操作过程

容易发现，在上述操作中，我们在插入操作中共使用了 $O(\log n)$ 次比较操作以及 1 次相等判定操作。而在删除操作中未使用任何操作。如果我们暴力实现上述过程，则每次操作所需的时间复杂度为 $\Theta(|S|)$，总的时间复杂度为 $\Theta(q^2)$，期望通过子任务 2。

操作类型	E-代价	C-代价	时间复杂度
插入	1	$\Theta(\log n)$	$\Theta(\|S\|)$
删除	0	0	
合计	$\Theta(q \log q)$	0	$\Theta(q^2)$

表 2: 算法 2

2.3 算法 3

在上述过程中，我们需要在线支持在任意位置插入一个元素、任意位置删除一个元素，并在该结构上进行二分。这是一个经典问题，可以通过若干方法解决，例如：

- 维护一棵平衡树，在二分时，我们只需要在平衡树上进行 $\Theta(\log |S|)$ 次查询第 k 大的操作，总的时间复杂度为 $\Theta(\log^2 |S|)$。
- 维护一棵平衡树，对于二分阶段，我们直接在平衡树上完成。总的时间复杂度为 $\Theta(\log |S|)$。

对于第一种方法，我们进行询问的次数即为在序列上进行二分查找的次数，可以保证在 $\lfloor \log_2 |S| \rfloor + O(1)$ 次比较操作内完成。对于第二种方法，则需要注意在询问部分带来的额外常数。需要注意的是，由于本题中限制了最劣情况下的操作次数，因此应当选择在最劣情况下树高仍有保证的平衡树来实现。

对于子任务 3（$Q \le 10^5$），实现上述任意算法（或其他可能的算法）均可通过。

操作类型	E-代价	C-代价	时间复杂度
插入	1	$\Theta(\log n)$	$\Theta(\log \lvert S\rvert)$ 或 $\Theta(\log^2 \lvert S\rvert)$
删除	0	0	
合计	$\Theta(q \log q)$	0	$\Theta(q \log q)$ 或 $\Theta(q \log^2 q)$

表 3: 算法 3

2.4 算法 4

对于子任务 $1 \sim 4$，我们只需要给出可能的候选解，即可得到 50% 的分数。

注意到在求解绝对众数问题时所使用的经典的**博耶-摩尔多数投票算法**（Boyer–Moore majority vote algorithm）[1]。该算法的伪代码如下：

Algorithm 1 博耶-摩尔多数投票算法

```
function Solve(X)
    n ← |X|
    v ← null
    c ← 0
    for i = 1 → n do
        if X_i = v then
            c ← c + 1
        else if c = 0 then
            v ← X_i
            c ← 1
        else
            c ← c − 1
        end if
    end for
    return v
end function
```

该算法的简要流程为：

- 维护一个候选值 v 以及其剩余的出现次数 c。
- 对于一个新的元素，如果它与候选值相同，则将其放入剩余元素中。
- 否则，候选值中的某个元素需要与该元素进行匹配，在匹配完成后，同时删除这两个元素。
- 特别地，若此时候选集合中已不包含任何元素，则该元素成为候选元素。

当绝对众数存在时，上述算法总是能够给出正确的绝对众数，无论序列中元素的分布顺序。我们考虑在进行插入与删除操作时维护 Boyer-Moore 多数投票算法的结果。

- 对于插入操作，由于 Boyer-Moore 算法的正确性不依赖插入元素的顺序，因此我们只需要运行上述算法即可。
- 对于删除操作，我们时刻维护出在算法过程中，每个元素所匹配的元素。在进行删除操作时：
  - 若该元素存在与其配对的某个元素，则我们只需要对与其配对的元素运行插入算法。
  - 否则，该元素必定属于当前局面下的候选集合，我们直接移除该元素即可。

下图展示了上述流程中的一个局面。

图 5: 插入 a_1

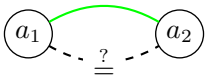

图 6: 插入 a_2

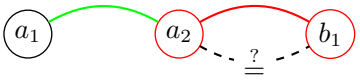

图 7: 插入 b_1

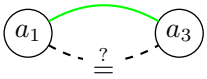

图 8: 插入 a_3

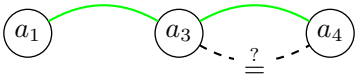

图 9: 插入 a_4

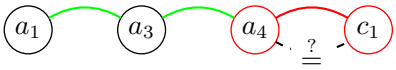

图 10: 插入 c_1

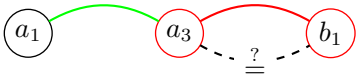

图 11: 删除 a_2，等价于插入 b_1

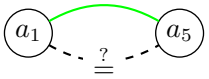

图 12: 插入 a_5

图 13: 删除 a_1，等价于直接移除

通过实现上述算法，我们即可得出一组候选解。不幸的是，由于我们无法快速检验一组解是否合法，因此我们只能给出候选解而不能确定众数的存在性。期望可以获得子任务 $1 \sim 4$ 中 50% 的分数。

操作类型	E-代价	C-代价	时间复杂度
插入	1	0	$\Theta(1)$
删除			
合计	1	0	$\Theta(1)$

表 4: 算法 4

值得注意的是，[2] 指出对于静态的绝对众数的存在性，在最坏情况下我们至少需要 $\left\lfloor \frac{3}{2}(n-1) \right\rfloor$ 次相等测试。

2.5 算法 5

对于子任务 5，只包含插入操作。我们考虑以下由 Gudmund Skovbjerg Frandsen 与 Sven Skyum 在 [3] 中提出的 tri-ladder 结构：

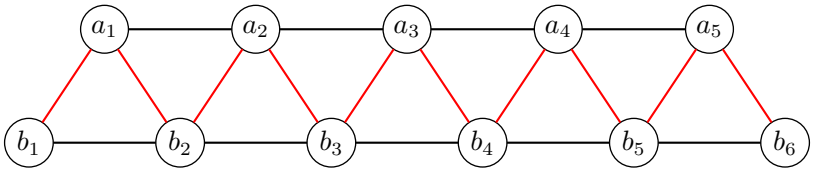

图 14: tri-ladder 结构

对于该结构，我们进行以下定义：

1. 整个结构由两行构成，分别为 (a) 行与 (b) 行。
2. 每个元素属于恰好一行。
3. 我们称两个元素**相邻**，当且仅当其为在同一行的一组元素 a_i, a_j（或 b_i, b_j），且 $|i-j|=1$。
4. 我们称两个元素**相对**，当且仅当其为在不同行的一组元素（不妨记为 a_i, b_j），且 $i=j$ 或 $i=j-1$。
5. 对于任意**相邻**的两个元素，我们已知他们的相等关系。
6. 对于任意**相对**的两个元素，我们保证他们不相等。
7. 若在当前局面下不存在绝对众数，则 (a) 行与 (b) 行的长度相差至多 1。

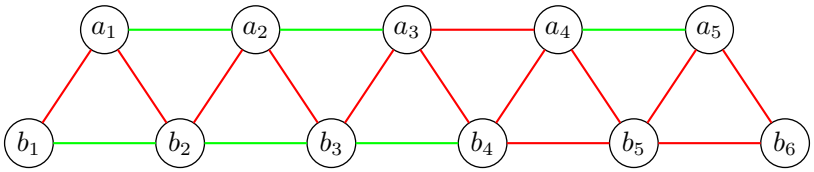

图 15: 一个不存在绝对众数的局面

8. 若在当前局面下存在绝对众数，则较长的一行（必定存在）的全体元素构成了唯一的绝对众数。

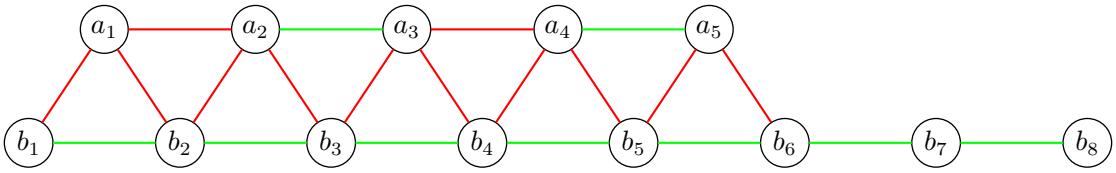

图 16: 一个存在绝对众数的局面

注意到，一旦我们可以快速维护该结构，我们便可快速维护出绝对众数及它的值。具体地，我们只需要进行如下判定：

- 若 (a) 行与 (b) 行长度相同，不存在绝对众数。
- 否则，则绝对众数存在当且仅当较长的一行仅包含一个由相等边构成的联通块。

我们不妨使用链表来维护所有极长的相等连续段。此时，我们便可以快速的支持定位一个连续段的开头，并在结尾追加或删除若干个元素。

对于插入操作，如果此时局面已经存在绝对众数（不妨设由 (b) 行元素构成），则我们的策略非常简单：

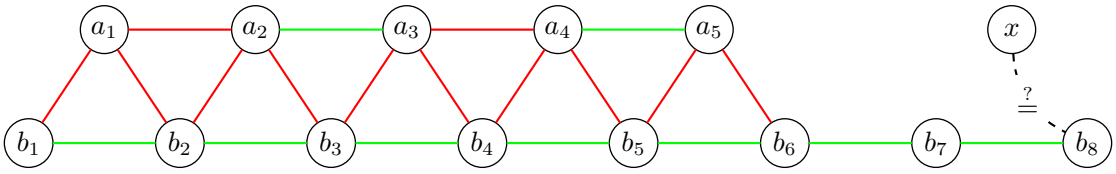

图 17: 在一个存在绝对众数的局面中执行插入操作

1. 若 x 与绝对众数相等，则直接将 x 插入绝对众数所在的行中。

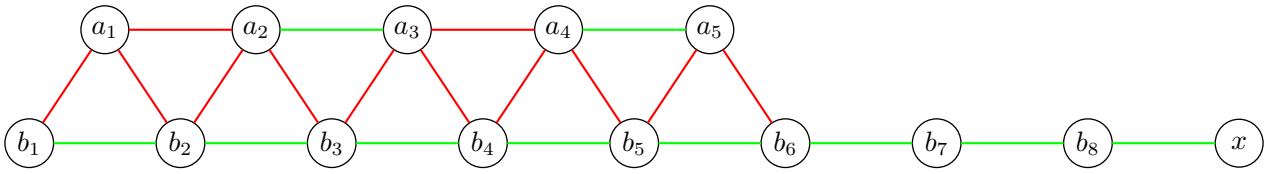

图 18: x 与绝对众数相等

2. 若 x 与绝对众数不相等，则直接将 x 插入另一行中。随后，需要执行额外的一次相等判定操作，来得到另一行结尾元素与 x 的相等关系，从而保证 tri-ladder 的结构。

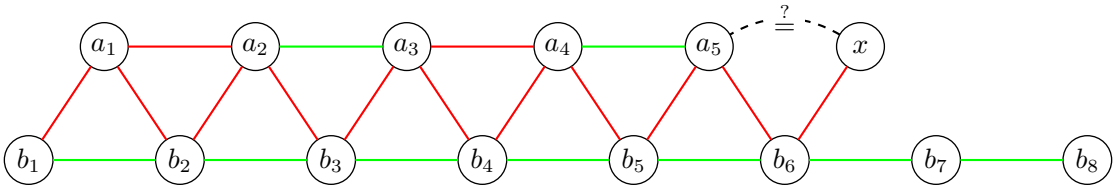

图 19: x 与绝对众数不相等

因此，在绝对众数存在的局面下，我们可以进行至多 2 次相等判定操作来插入一个元素。

当绝对众数不存在时，不妨设此时 (b) 行的长度大于 (a) 行。记 (a) 行的元素为 $a_1, a_2, \cdots, a_n$，(b) 行的元素为 $b_1, b_2, \cdots, b_m$，且 $m = n + 1$（对于 $m = n$ 的情形，插入操作是类似的）。对于新加入的元素 x，我们首先询问 x 与 b_m 是否相等：

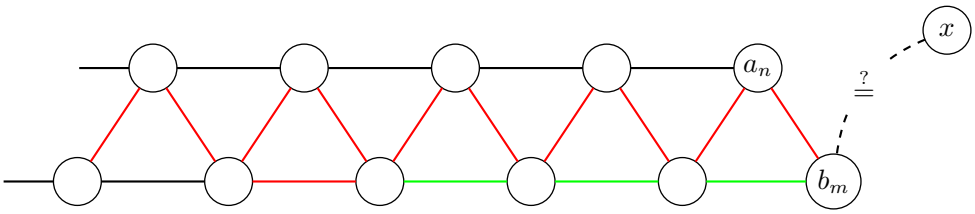

图 20: 插入元素 x (1)：询问 x 与 b_m 是否相等

1. 若 $x \neq b_m$，由于 (b) 所在行元素个数大于等于 (a) 所在行元素个数，因此无论绝对众数存在与否，我们均可将 x 插入 (a) 所在行中。在插入后，为了保证 tri-ladder 的性质，我们需要询问 a_n 于 x 的关系。

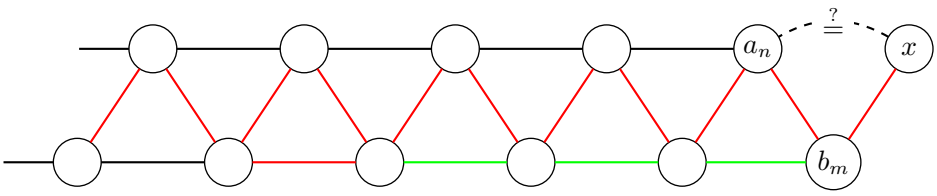

图 21: 插入元素 x (2)A：询问 x 与 a_n 是否相等

2. 否则，我们尝试将 x 置入我们的结构。记 b_m 所在的相等连续段的最左端节点的编号为 b_l，其左侧的点为 b_o。同时与 b_l, b_o 相对的点为 a_l，a_l 前第一个点为 a_o，如下图所示：

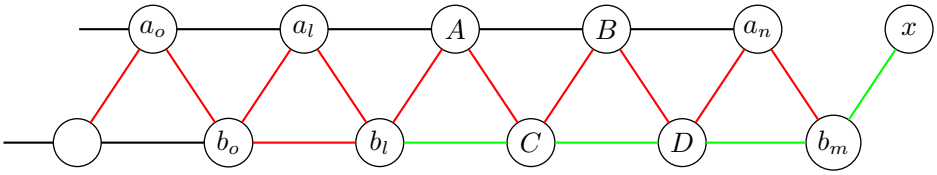

图 22: 点的标号

(a) 需要注意的是，由于绝对众数不存在，因此我们增加辅助元素不会对我们的结构造成影响。具体地，在上述顶点中，如果对应的顶点不存在，我们可以建立虚点 O 代替，并定义 $\forall x \in D$ 总是有 $x \neq O$。

(b) 若 $a_o \neq a_l$，则我们将行 (a) 中的顶点 $a_l \cdots a_n$ 与行 (b) 中的顶点 $b_l \cdots b_m$ 两两交换，得到下图所示的结构。容易发现，我们唯一无法确定的即为 a_o 与 b_l 的关系，因此我们进行额外的一次询问来确定他们的关系。而我们唯一拆散的结构为 a_o 与 a_l 所在的块，但又由 $a_o \neq a_l$，可知我们没有破坏任何一个原有的块，因此我们仍然可以用链表/并查集来维护每个块开头的信息。

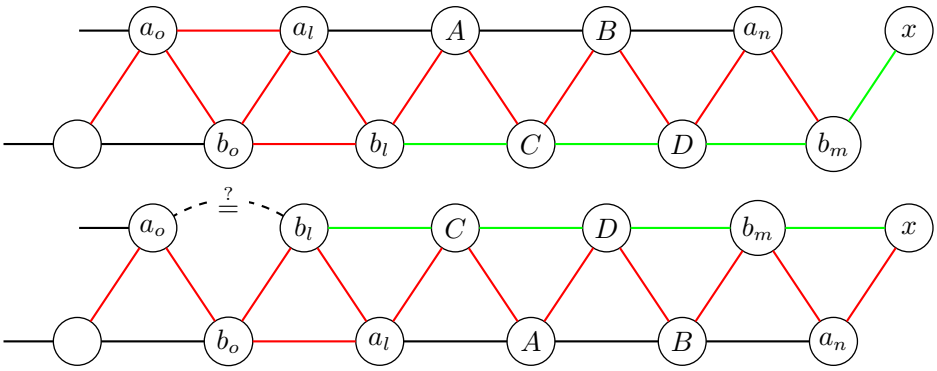

图 23: 插入元素 x (2)B(a)

(c) 否则，记 b_o 前的元素为 b_t，并考虑交换 x 与 b_o 的位置。由于 $a_o = a_l$，且 $a_l \neq b_l$，而 $b_l = x$，因此必有 $x \neq a_o$ 且 $x \neq a_l$，满足 tri-ladder 的限制。在操作完成后，我们新增了相邻点对 (b_t, x) 与 (a_n, b_o)，因此需要分别进行相等判定操作。

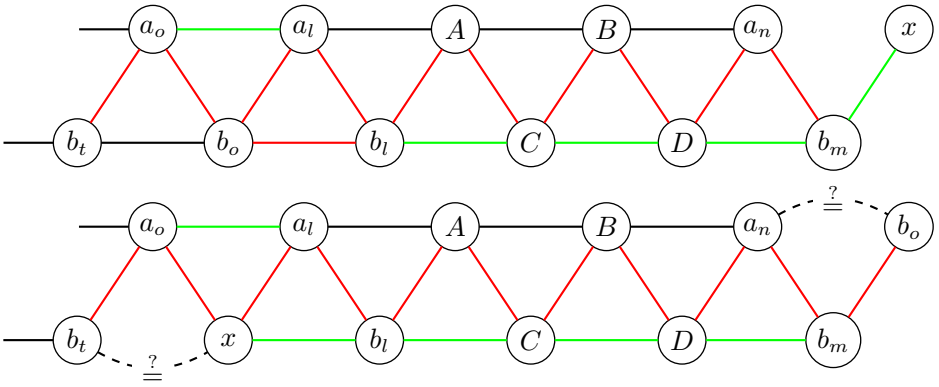

图 24: 插入元素 x (2)B(b)

综上，我们可以在最坏进行 3 次相等判定的情况下，支持在此结构中插入一个元素。

操作类型	E-代价	C-代价	时间复杂度
插入	3	0	$\Theta(1)$
删除	/	/	/
合计	3	0	$\Theta(q)$

表 5: 算法 5

2.6 算法 6

我们考虑扩展算法 5 中的 tri-ladder 结构，使其支持删除操作。
对于绝对众数存在的情形，删除操作是容易的：

1. 若删除的元素 x 非绝对众数，则直接将对应的元素删除，并将后续顶点全都前移一位即可。操作完后，需要进行额外的一次相等判定操作。

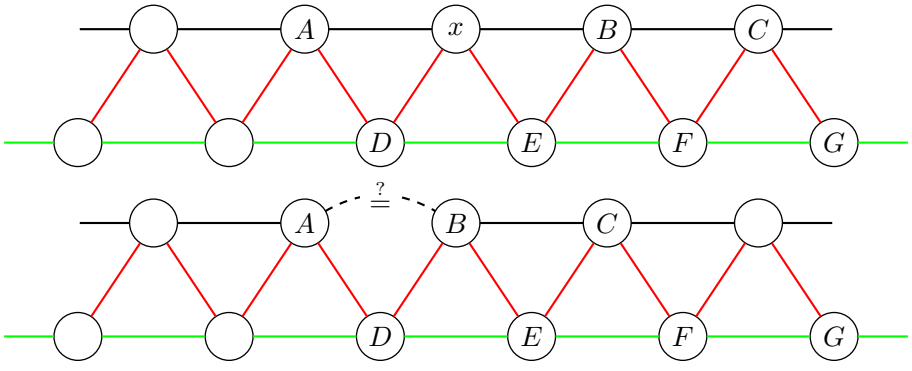

图 25: 删除元素 x：x 非绝对众数

2. 若删除的元素为绝对众数，同样可进行上述操作，并需要额外判定此时是否绝对众数仍存在。

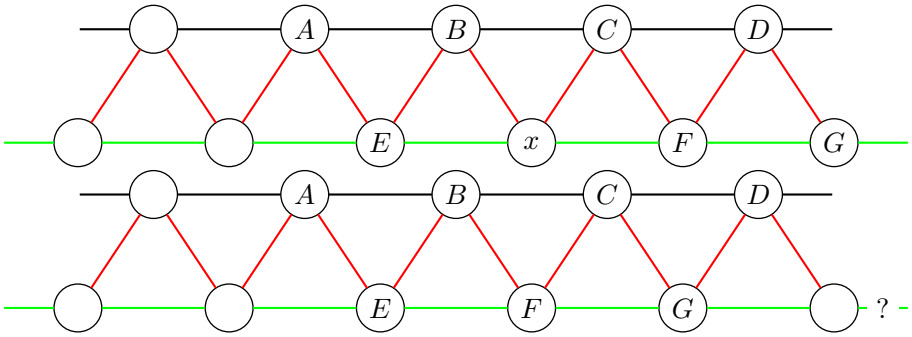

图 26: 删除元素 x：x 为绝对众数

否则，同样地，不失一般性，我们假设删除操作对应 (a) 行中的某一元素 x，并假设其前一个元素为 a_l，后一个元素为 a_r。x 与 a_l 相对的元素为 b_l，x 与 a_r 相对的元素为 b_r。

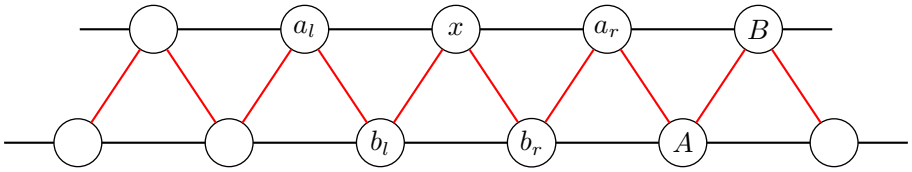

图 27: 删除元素 x：节点的标号

首先，最平凡的策略为，在移除 x 后，将 x 右侧的所有节点向前补一位，即将节点 b_r 移动至节点 x，节点 a_r 移动至节点 b_r，节点 A 移动至节点 $a_r\cdots\cdots$。我们注意到，对于相对的元素，只有 b_l 与 b_r 成为了新的相对的元素，因此我们唯一需要满足的条件即为 $b_l \neq b_r$。

而当 $b_l = b_r$ 时，我们有以下朴素策略：同时移除顶点 x 与 b_r，并重新将 b_r 插入结构中。为了使得整个结构仍然可以使用链表快速维护，我们需要按照 a_l 与 x 是否为某一连续段的非边界点进行分类：

1. 若 $a_l \neq x$ 且 $b_l \neq b_r$，则我们可以直接移除顶点 x，并进行上述操作。在上述操作后，我们需要更新 a_l 与 b_r 的关系和 b_l 与 a_r 的关系，进行 2 次相等判定操作：

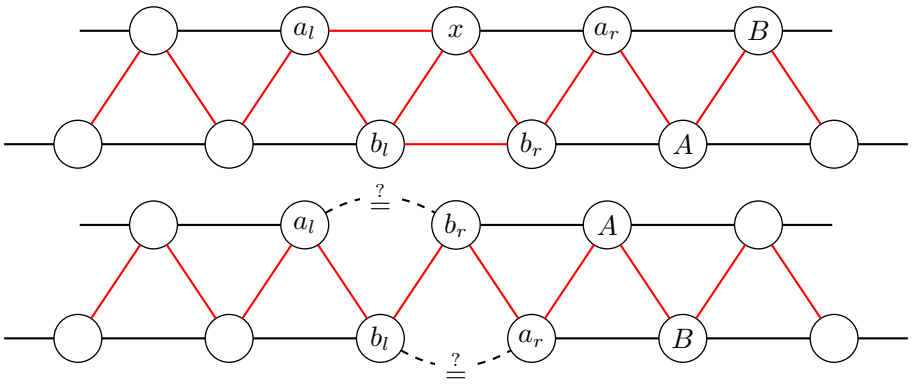

图 28: 删除元素 x：$a_l \neq x$ 且 $b_l \neq b_r$

2. 若 $b_l = b_r$，则我们同时移除 x 与 b_r，并对 b_r 重新执行插入操作。注意到 $b_l \overset{?}{=} A$ 总是已知，因此我们只需要额外进行 1 次相等判定操作。

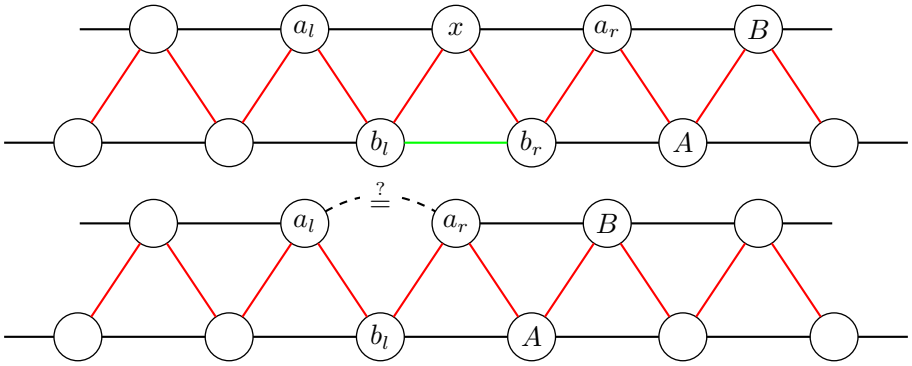

图 29: 删除元素 x：$b_l = b_r$

3. 若 $a_l = x$，则我们同时移除 x 与 b_l，并对 b_l 重新执行插入操作。同样注意到 $a_l \stackrel{?}{=} a_r$ 总是已知，因此我们只需要额外进行 1 次相等判定操作。

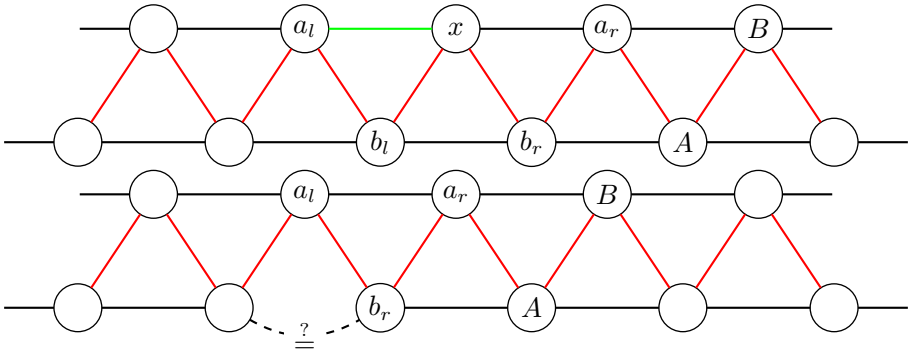

图 30: 删除元素 x：$a_l = x$

对于第一种情形，我们需要使用 2 次相等判定操作。对于第二种情形，我们需要 1 次相等判定操作与 1 次插入操作，因此在最坏情况下删除操作所需 4 次相等判定操作。

操作类型	E-代价	C-代价	时间复杂度
插入	3	0	$\Theta(1)$
删除	4		
合计	4	0	$\Theta(q)$

表 6: 算法 6

2.7 算法 7

算法 6 的瓶颈在于，我们使用链表维护了每个相等边所对应的连续段，导致我们不能快速的在任意位置分裂与合并。事实上，我们可以使用平衡树来维护所有的连续段，来支持快速的定位、分裂与合并。

首先着手于解决插入操作。

1. 对于 $x \neq b_m$，不需要进行任何处理。

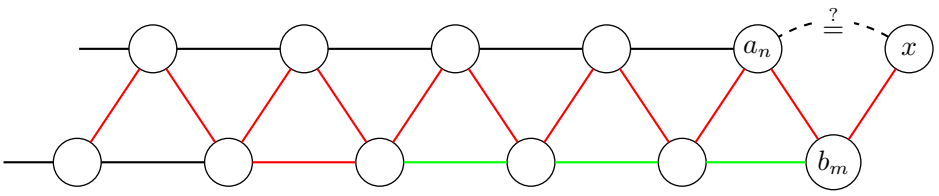

图 31: 更优地插入元素 x (2)A：询问 x 与 a_n 是否相等，无特殊处理

2. 无论何种情况，我们都采取与原本 $a_o \neq a_l$ 一样的操作，将 $a_l \cdots a_n$ 与 $b_l \cdots b_n$ 进行两两交换。由于我们使用平衡树维护所有极长连续段，因此无需担心在中间断开某个连续段的情形。为了实现的方便，我们可以将 x 提前至该连续段的开头，这样在交换完成后，所有点所相对的点不会改变。

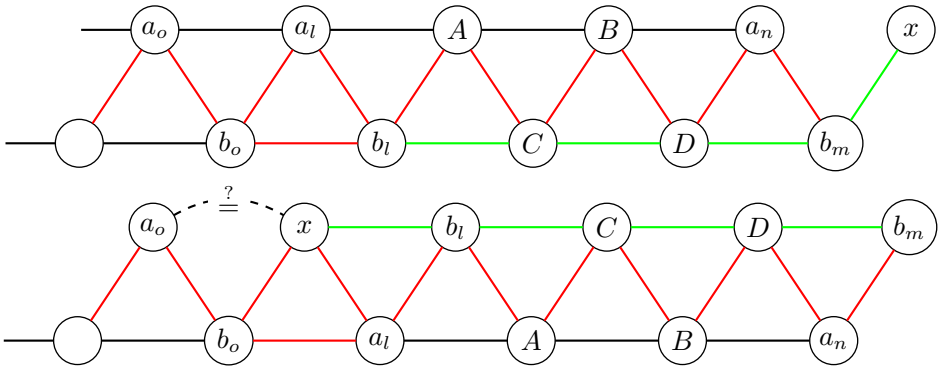

图 32: 更优地插入元素 x (2)B

综上，插入操作所需的比较次数被优化至 2 次，同时删除操作所需的比较次数被优化至 3 次。此算法可完整通过子任务 5，并得到子任务 6 中除最后一档评分规则外的所有分值。

操作类型	E-代价	C-代价	时间复杂度
插入	2	0	$\Theta(\log \lvert S \rvert)$
删除	3		
合计	3	0	$\Theta(q \log q)$

表 7: 算法 7

2.8 算法 8

事实上，我们还可以进一步优化删除操作。我们类比对插入操作的处理：

1. 若 $b_l = b_r$，记 b_l 所在的由相等的边构成的连续段中最左侧的点的编号为 b_u，并对应找到与其相对的且最靠左的顶点 b_v，如下图所示。在删除顶点 x 后，只需与插入部分类似，将连续段 $b_u \cdots b_r$ 移动到另一侧，并将结尾空位补齐即可。

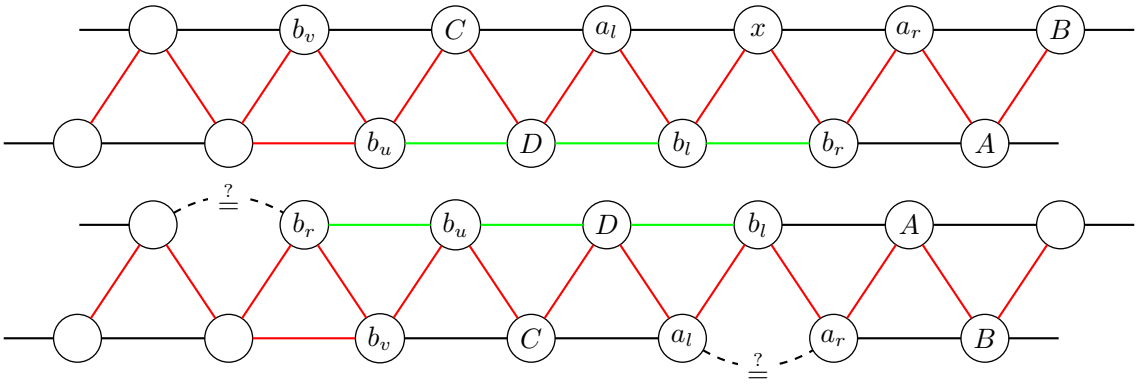

图 33: 更优地删除元素 x：$b_l = b_r$

2. 否则，我们在删除顶点 x 后，直接将在 x 后方的节点向前补齐即可，并需要额外补齐 a_l 与 b_r 和 a_r 与 b_l 的信息，同样进行 2 次相等判定操作即可。

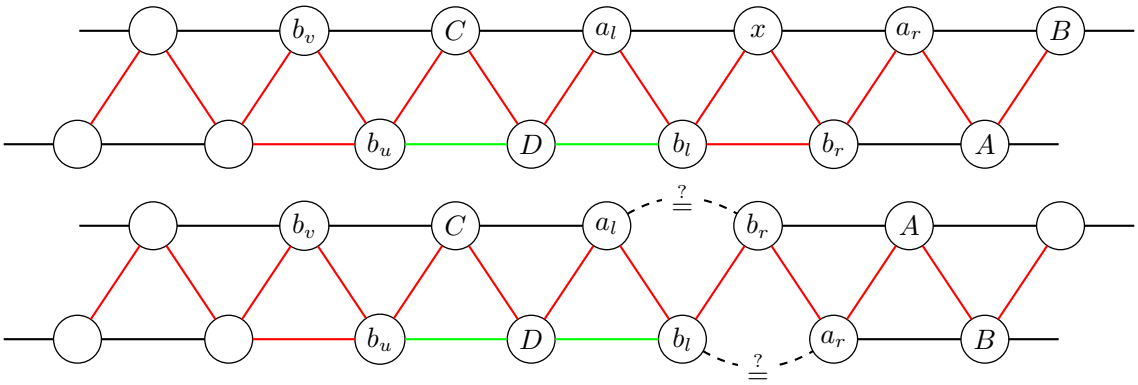

图 34: 更优地删除元素 x：$b_l \neq b_r$

综上，在使用平衡树快速维护连续段的起点时，由于不需要调整到边界情况，因此只需要 2 次相等判定，即可完成删除操作。

操作类型	E-代价	C-代价	时间复杂度
插入	2	0	$\Theta(\log \lvert S \rvert)$
删除			
合计	2	0	$\Theta(q \log q)$

表 8: 算法 8

期望可以通过所有子任务，获得满分。

3 备注

本题的准备过程中，同来自山东省潍坊第一中学的刘一平（集训队）、来自中国人民大学附属中学的黄洛天、来自北京大学的彭博、来自山东省平邑第一中学的唐绍轩与来自浙江省诸暨市海亮高级中学的王鸿翼进行了交流。

参考文献

[1] Robert S Boyer and J Strother Moore. Mjrty—a fast majority vote algorithm. In Automated Reasoning, pages 105–117. Springer, 1991.

[2] Michael J Fischer and Steven L Salzberg. Finding a majority among n votes. Technical report, YALE UNIV NEW HAVEN CT DEPT OF COMPUTER SCIENCE, 1982.

[3] Gudmund S Frandsen and Sven Skyum. Dynamic maintenance of majority information in constant time per update. Information processing letters, 63(2):75–78, 1997.