可持久化数据结构

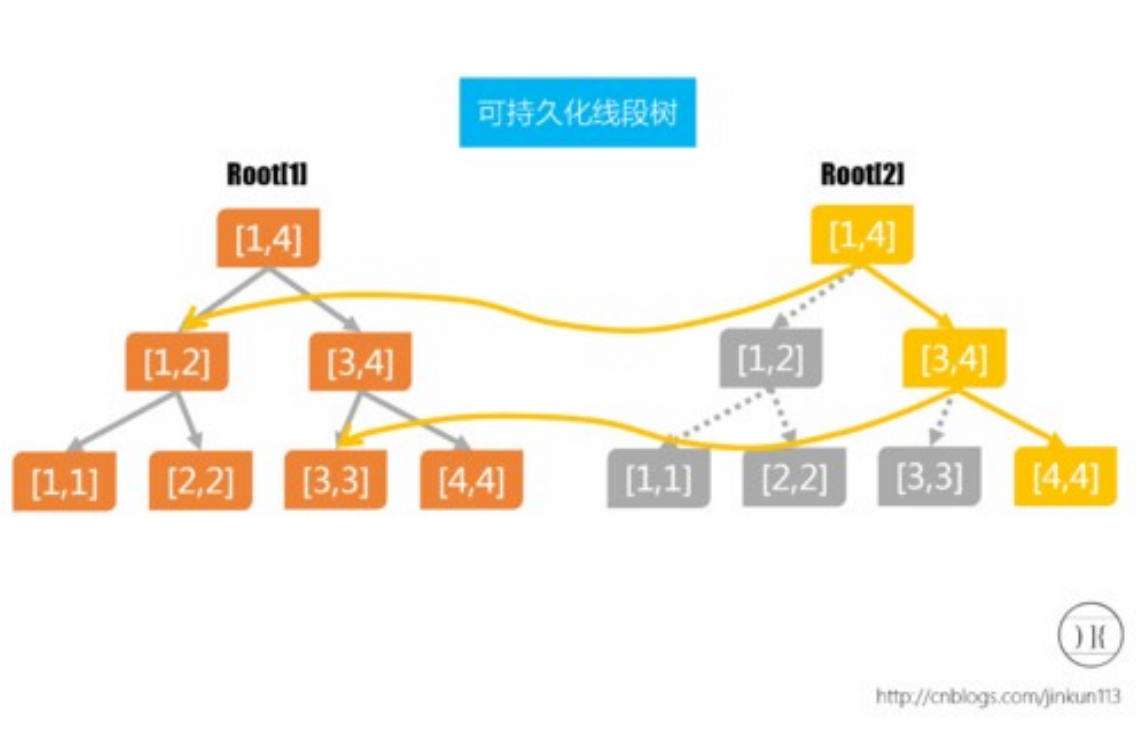
成都七中　nzhtl1477

可持久化的定义

- 我们对于一个数据结构，想维护其所有的历史版本
- 部分可持久化：允许访问历史版本，不能修改历史版本
- 完全可持久化：允许把目前版本替换为历史版本
- Confluent persistent：允许合并历史版本，几乎不太能维护

可持久化线段树

- 如果每次都暴力复制整棵线段树，时间复杂度无法承受
- 考虑每次修改，线段树只有 O(logn) 个节点会被修改，所以我们可以对每个被修改的节点，将其复制一份进行修改



可持久化线段树

- 可以发现这样做，我们时间复杂度和空间复杂度都变成了 O(mlogn)，比起暴力复制改善了很多
- 很多情况下，如果题目没有强制在线，那么我们可以用分治来解决，而不用可持久化

经典问题

- 给一个序列，每次查询区间的 k 小值

值域线段树

- 定义值域线段树：我们在离散的值域上建立一棵线段树，可以考虑预先进行离散化来缩小值域，这时线段树每个节点表示值在一个范围内的数，在这个例子中我们只需要统计一个范围内有多少数。

Solution

- 我们先维护出每个前缀的值域线段树，这个可以用刚才讲的可持久化的方法
- 每次从前缀 [1,t] 拓展到 [1,t+1] 的时候，即在 t 版本的值域线段树上插入 a[t+1] 这个值，并将这个版本记做第 t+1 个版本
- 发现可以用之前讲的技巧进行优化，使得复杂度做到 O(nlogn)

Solution1

- 我们维护出了每个前缀的值域线段树，考虑怎么求出 kth
- 发现可以二分答案，然后转化为区间中小于 x 的数个数
- 我们可以用 r 位置和 l-1 位置的值域线段树差分，即在两个历史版本的线段树上都查询小于 x 的数个数，来知道区间小于 x 的数个数
- 总时间复杂度 O(nlogn + mlog^2n)

Solution2

- 在一棵值域线段树上，我们可以可以二分来求出 kth
- 由于值域线段树结构是相同的，所以可以很方便地一起二分
- 每次传两个指针，表示当前走到的 r 位置树的节点 a ，和当前走到的 l-1 位置树的节点 b
- 每次二分的时候，判断当前的 k 和 a -> left -> size - b -> left -> size 大小即可，和普通的线段树上一起二分类似

- 总时间复杂度 O((n+m)logn)
- 存在 O((n+m)logn/loglogn) 的解法

Path Copy

- 上述的方法即 path copy，字面意思就是每次复制被修改的整条路径来实现可持久化
- 在竞赛中，一般来说 path copy 的功能是足够的
- 优点：方便理解，出题人也只会这个
- 缺点：空间有时候显得很大

Fat Node

- 我们每个节点开一个 vector，如果这个节点被访问到了，则在这个节点的 vector 中 push_back 一个二元组信息，表示这个节点在某个时刻被更改为了一个状态
- 每次我们访问到一个节点的信息时，则在其 vector 上二分（不一定要这么做，vector 可以换成其他数据结构，这里是为了简单理解），找到距离查询的时刻最近的一次修改之后这个节点的信息变成了什么，然后返回结果

Fat Node

- 可以发现这样修改复杂度 O(logn)，如果使用 vector，则查询复杂度为 O(log^2n)，这个复杂度是难以优化的，目前没有方法优化到 O(logn) 的复杂度，但是可以做到 O(logn loglogn)
- 优点：修改高效，空间常数小
- 缺点：查询低效

Note

- 还有其他很多可持久化的技术，算法竞赛圈在可持久化这个方面感觉内容很少

HDU - 4757 Tree

- 给出一颗树，每棵树上有 n 个结点，每个结点对应一个值，有m组查询操作，查询在从 x 到 y 的这条路径上选出一个数与 z 进行异或的最大值。
- 值域是 n

Solution

- 考虑如何类比前面的区间 kth 做法

Solution

- 给定一个集合，如何查询一个数 x 与集合的最大 xor 和?
- 对集合建立一棵可持久化 01Trie （和值域线段树一样）
- 从高位到低位贪心，如果当前 x 这一位是 0，则尽可能走 1，否则尽可能走 0

Solution

- 树上怎么做?
- 考虑树上前缀和的方法，从根到叶子建立可持久化 01Trie
- 每次查询的时候可以在 O(1) 棵 Trie 上一起二分实现，和区间 kth 是类似的

- 复杂度 O((n + m)w)

Notice

- 如果值域很大怎么做?
- 比如值域是 10^9？
- 之前的区间 kth 是可以离散化的，但是 xor 和所需要的可持久化 trie 是不能离散化的，因为这里值是有意义的，离散化只是保留了一个大小关系
- 但是又不能 O(值域) 复杂度把这个树直接建出来

动态开点

- 使用动态开点的方法可以有效减少空间浪费
- 可以发现每次修改只会访问到 $O(\log v)$ 个节点
- 我们不需要把线段树预先建出来，直到访问到一个节点的时候，如果其为空，再建出来
- 这样每次只需要建立 $O(\log v)$ 个节点，空间复杂度得以改善

Bzoj3524 [Poi2014]Couriers

- 给一个长度为 n 的序列 a 。 1≤a[i]≤n 。
  m 组询问，每次询问一个区间 [l,r] ，是否存在一个数在 [l,r] 中出现的次数大于 (r-l+1)/2 。
- 如果存在，输出这个数，否则输出 0 。

Solution1

- 对序列建立可持久化 Trie，每次查询的时候可以类比区间 kth：
- a 对应了 [1,l-1] 的 Trie， b 对应了 [1,r] 的 Trie
- 如果 a 的左子树大小 -b 的左子树大小 >= 区间长度一半，则答案可能在这里
- 如果 a 的右子树大小 -b 的右子树大小 >= 区间长度一半，则答案可能在这里
- 否则答案不存在

- O((n+m)logn)

Solution2

- 考虑随机化，每次随机区间中一个位置，然后查这个位置在区间中的出现次数
- 如果满足条件，则找到了答案
- 由于答案的出现次数 >= 区间长度一半，所以每次随机到的概率是一个常数，随机 log 次的话大概错误率是有保证的（没仔细算但我记得是对的）

- O(mlog^2n)

Bzoj2653: middle

- 一个长度为 n 的序列 a，设其排过序之后为 b，其中位数定义为 b[n/2]，其中 a,b 从 0 开始标号，除法取下整。
- 给你一个长度为 n 的序列 s。
  回答 Q 个这样的询问：s 的左端点在 [a,b] 之间，右端点在 [c,d] 之间的连续子序列中，最大的中位数。其中 a<b<c<d。
- 强制在线

Solution

- 这种形式的查询如何处理呢?
- 不难发现实际上就是我们必须选出 [b,c] 这个区间
- 然后两边可以各增加一个 [l,b) 和 (c,r]，其中 l 在 [a,b] 中，r 在 [c,d] 中

Solution

- 中位数的通用处理方法?
- 二分答案 x，把小于 x 的设为 -1，大于等于 x 的设为 1
- 如果区间和 >=0，则 >=x 的数出现次数 >= 区间长度的一半，则中位数 >=x

Solution

- 于是我们可以二分答案 x
- 验证的时候，先查询 [b,c] 中 >=x 的数个数
- 然后需要找到左右两边和最大的一段?
- 从小到大加入权值，把小于当前权值的设为 -1 ，否则设为 1
- 只需要查询加入权值为 x 的时候的线段树：
- 用线段树维护区间和与最大的前后缀和，这样就可以找到一个点向两边最大延伸出的一段了
- 可持久化这个线段树即可
- O(nlogn + mlog^2n)

【UNR #1】火车管理

- 有 n 个栈，m 次操作。
- 1.将 x 压入 [l,r] 的栈中
- 2.将 l 的栈顶弹出
- 3.询问 [l,r] 栈顶的和

- n,m≤5×10^5

Solution

- 第一个和第三个操作都是基本的线段树操作，第二个操作我们可以把它看做返回该元素的上一个版本，不难想到用可持久化线段树去维护。
- 我们在可持久化线段树上维护两个信息，当前节点的版本和当前区间的权值，这个版本标记我们可以把它看做 lazy 标记，每次访问的时候下放即可。

- O(mlogn)

可持久化数组

- 字面意思，我们需要完全可持久化一个数组，这里强制在线
- 在 OI 中，我们使用可持久化线段树来实现可持久化数组，时间复杂度 O(logn)
- 这个问题被证明了复杂度为 θ(loglogn)，因为使用 vEB 树或 y-fast trie 而 not practical，具体做法不做详细介绍

Luogu3402 可持久化并查集

- 1 a b 合并 a,b 所在集合
  2 k 回到第 k 次操作之后的状态（查询算作操作）
  3 a b 询问 a,b 是否属于同一集合，是则输出 1 否则输出 0
  请注意本题采用强制在线，所给的 a,b,k 均经过加密，加密方法为 x = x xor lastans,lastans 的初始值为 0

可持久化并查集

- 可以使用可持久化数组来实现可持久化并查集
- 使用按秩合并的并查集，每次操作时需要访问 O(logn) 个数组位置
- 数组使用可持久化线段树来实现即可

- 时间复杂度 O(log^2n)
- 存在 O(logn) 的实现，但是在 OI 中没有价值

可持久化平衡树

- 我们的平衡树也是可以可持久化的
- 很多人关于可持久化平衡树都在瞎扯
- 1. 带旋转的平衡树不能可持久化——错的
- 2. 维护了父亲的平衡树不能可持久化——错的
- 3. 可持久化 treap 功能是足够的——错的，至少我们不知道怎么做

可持久化平衡树

- 如果是插入删除版本的，则只需要在修改的路径上复制一遍即可，旋转的时候也需要复制一遍
- 为了方便，请自顶向下维护，并且不维护父亲节点
- 虽然维护父亲节点也可以无复杂度代价的可持久化，但是很复杂，而且常数很大，所以还是别维护了

可持久化平衡树

- 如果是分裂合并版本的，则分裂合并的时候新建节点即可，写起来和不可持久化的平衡树差不多，很方便
- 具体如何实现这里就不详细介绍了

可持久化平衡树能做什么

- 可以发现可持久化平衡树维护权值的话一般是不如可持久化 Trie 的，因为后者好写，而且常数更小，而且复杂度也是 O(w)= O(logn) 的
- 感觉最大的一个应用是做区间复制，比如把一个区间复制到另一个区间上这样的操作

可持久化平衡树能做什么

- 这样的操作有很大的局限性，因为我们平衡树的复杂度是 O(logn)，而一次这样的操作就会使得 n 扩大一倍，所以到最后 n 会变成指数级大小，导致平衡树复杂度退化为 O(n)，这里还不考虑高精度造成的影响
- 虽然合并两个任意大小的平衡树可以做到 O(1)，但一旦涉及到修改，复杂度还是会炸掉
- 所以一般会限定序列长度

可持久化 treap 的缺点

- 需要注意的一点是，由于 treap 随机种子的问题，导致其在可持久化的时候会出一些问题
- 比如对一个节点进行自我复制的时候，会导致复制出来的节点和原节点的随机种子相同，这样复杂度会出错
- 如果按照 size 随机进行旋转，在一些特定的问题中也会出错，比如自我复制 k 次时复杂度会多 log

HDU 6087 Rikka with Sequence

As we know, Rikka is poor at math. Yuta is worrying about this situation, so he gives Rikka some math tasks to practice. There is one of them:

Yuta has an array A with n numbers and he keeps a copy of the initial value of array A as $A'(A'_i = A_i)$. Then he makes m operations on it.

There are three types of operations:
1 l r: Yuta wants Rikka to sum up A_i for all i in $[l, r]$.
2 l r k: Yuta runs the C++ code "for (int i=l;i<=r;i++) A[i]=A[i-k]" on sequence A. (The pascal version of the code snippet is "for i:=l to r do A[i]:=A[i-k]").
3 l r: Yuta changes A_i back to A'_i, for all $i \in [l, r]$.

It is too difficult for Rikka. Can you help her?

Solution

- 区间复制 k 次如何解决?
- 可以倍增地复制，只要合并两个大小为 n 和 m 的节点复杂度是 O(log(n/m)) 就可以保证复杂度
- 还有一种做法是类似之前讲的平衡树维护连续段
- 我们可以将区间复制也看做是一次类区间染色操作
- 每个平衡树节点额外维护出其是否是一次区间复制产生的，并且是由哪个节点复制了多少次
- 分裂的时候特判这种节点即可

Solution1

- 这样直接做空间是 O(mlogn) 的，还带大常数
- 可以使用 smart pointer 来减少内存
- Smart pointer：对每个节点开一个额外的域，表示有多少个指针指向其，如果一个点没有被指针指向，就将其回收，在没有环的图上这个方法是可以保证空间复杂度的，而可持久化的过程是一个 DAG 结构，所以可以保证回收掉没有意义的节点
- 有些人说这个方法常数大，我感觉额外常数应该很小
- stl 的话这里会多访存一倍，所以比较慢，如果可以内嵌的写的话不会多访存

Solution2

- 可以每 θ(n/logn) 次重构一次整个序列，并且回收内存
- 发现这样空间是： θ(n/logn) * O(logn) = O(n)
- 时间是 m / θ(n/logn) * n = θ(mlogn)

- 时间复杂度 O(mlogn)，空间复杂度 O(n + m)

可持久化可合并堆

- 1. 插入
- 2. 删除最小值
- 3. 查询最小值
- 4. 合并两个堆
- 目前最优复杂度是除了 2 操作 pop 复杂度为 O(logn) 以外，其他操作复杂度均为 O(1)
- 可以完全可持久化，并且一定程度上可以合并历史版本

可持久化可合并堆

Brodal Queue

- 这是什么？
- http://en.wikipedia.org/wiki/Brodal_queue
- 插入、查询最小值、合并、减小某个元素的值都是最坏复杂度 $O(1)$ 的，只有删除操作是 $O(\log n)$ 的。
- 并且它能够可持久化。

- 网上流传的一个关于可持久化可并堆的 pdf
- 这个说法也是错的，可持久化的 Brodal Queue 无法减小某个元素的值，只能做上述的四个操作，而且 Brodal 论文中给出的是一个新的数据结构，叫做斜二项堆
- 如果对 Brodal queue 感兴趣可以看 https://www.luogu.com.cn/problem/P6019

可持久化可合并堆

- 斜二项堆在除了 decrase key 以外可以做到最优理论复杂度，并且代码非常简单，效率也不差
- 但是一般用可持久化的左偏树就够用了，可持久化可并堆本来也没什么题

Luogu3293 [SCOI2016] 美味

- 一家餐厅有 n 道菜，编号 1...n ，大家对第 i 道菜的评价值为 ai(1<=i<=n)。有 m 位顾客，第 i 位顾客的期望值为 bi，而他的偏好值为 xi 。因此，第 i 位顾客认为第 j 道菜的美味度为 bi XOR (aj+xi)，XOR 表示异或运算。
- 第 i 位顾客希望从这些菜中挑出他认为最美味的菜，即美味值最大的菜，但由于价格等因素，他只能从第 li 道到第 ri 道中选择。请你帮助他们找出最美味的菜。

Solution

- 按照数位一位一位贪心，因为整体加了一个 x ，所以我们考虑对于所有的 (ai+x) 与 b 的按位异或。

Solution

- 还是考虑按位贪心，不过这里每个数都加上了一个数，所以会复杂一些
- 假设我们已经处理到 b 二进制下的第 i 位，假设是 1 （0 同理），
- 那么我们只需要查找是否存在 aj+x 使得其二进制第 i 位数字是 0，我们已经处理了前 i-1 位了，设当前结果是 ans，那么我们需要查找的数的大小就是区间 [ans-x,ans+(1<<i)-1-x]
- 那么现在我们就是要在 a[l]...a[r] 中找出是否存在于 [ans-x,ans+(1<<i)-1-x] 的数字，可以使用区间小于 x 的数个数来做
- 总时间复杂度 O(nlogn + mlognlogv)，可以除 polylog logn

某经典问题

- 给一个 DAG，每个边有边权，每次查询从一个点出发到任意点，字典序 kth 的路径，是到哪个点
- n,m<=1e5,k<=1e18
- 每个点的每种边权的出边只有一条
- 强制在线

Solution

- 每个节点用一个可持久化平衡树维护答案
  每次在 DAG 上按拓扑顺序合并一个点到的所有边的平衡树，每个平衡树是按原顺序合并过来的
- 然后这个合并的时候取前 maxk 个节点就行了，不然是指数级的

Bzoj3551 Peaks 加强版

- 有 n 个座山，其高度为 hi 。有 m 条带权双向边连接某些山。多次询问，每次询问从 v 出发 只经过边权 <=x 的边 所能到达的山中，第 K 高的是多少。
- 强制在线

Solution

- 我们把平衡树启发式合并或者 trie 合并的过程可持久化一下
- 注意到这个可持久化是部分可持久化，只需要访问历史版本，不需要在历史版本上面修改，所以复杂度是不变的

- 看具体实现，复杂度为 O((n+m)logn) 或 O((n+m)log^2 n)

某经典问题

- 给很多模式字符串，每次查询时给两个字符串 a,b ，问有多少模式字符串前缀是 a ，后缀是 b
- 字符串总长度 10^6 ，字符串和询问个数 10^5

Solution

- 考虑开两棵 trie 树，分别把所有模式字符串顺序和倒序插入
- 这样我们查询时也将 a 串顺序在 trie 上跑， b 串倒序在 trie 上跑
- 问题转换为在第一棵 trie 树的子树中和第二棵 trie 树的子树中有多少共同元素
- 类似之前树套树中讲的，将每个点在第一棵树 DFS 序位置当做 x 坐标，在第二棵树 DFS 序位置当做 y 坐标，转换为二维数点

- O(|S|+(n+m)logn)

Luogu4899 [IOI2018] werewolf 狼人

题目背景

展开

本题为交互题，但在此请提交**完整程序**。

题目描述

在日本的茨城县内共有 N 个城市和 M 条道路。这些城市是根据人口数量的升序排列的，依次编号为 0 到 $N-1$。每条道路连接两个不同的城市，并且可以双向通行。由这些道路，你能从任意一个城市到另外任意一个城市。

你计划了 Q 个行程，这些行程分别编号为 0 至 $Q-1$。第 $i(0 \leq i \leq Q-1)$ 个行程是从城市 S_i 到城市 E_i。

你是一个狼人。你有两种形态：**人形**和**狼形**。在每个行程开始的时候，你是人形。在每个行程结束的时候，你必须是狼形。在行程中，你必须要变身（从人形变成狼形）恰好一次，而且只能在某个城市内（包括可能是在 S_i 或 E_i 内）变身。

狼人的生活并不容易。当你是人形时，你必须避开人少的城市，而当你是狼形时，你必须避开人多的城市。对于每一次行程 $i(0 \leq i \leq Q-1)$，都有两个阈值 L_i 和 $R_i(0 \leq L_i \leq R_i \leq N-1)$，用以表示哪些城市必须要避开。准确地说，当你是人形时，你必须避开城市 $0, 1, \ldots, L_i-1$；而当你是狼形时，则必须避开城市 $R_i+1, R_i+2, \ldots, N-1$。这就是说，在行程 i 中，你必须在城市 $L_i, L_i+1, \ldots, R_i$ 中的其中一个城市内变身。

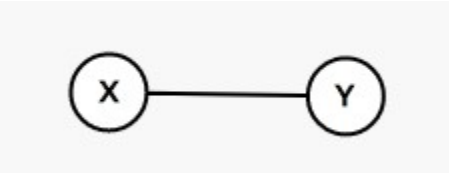
你的任务是，对每一次行程，判定是否有可能在满足上述限制的前提下，由城市 S_i 走到城市 E_i。你的路线可以有任意长度。

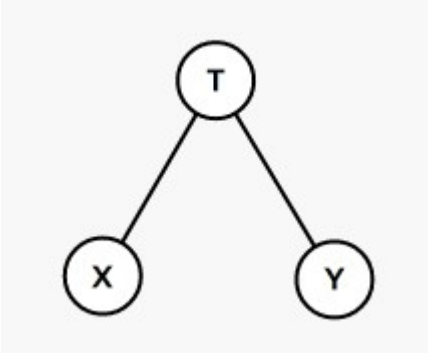
Solution

- 题意即是否存在一条 (si, ti) 的路径，满足先只走编号不超过 Li 的点，再走编号超过 Ri 的点

Kruskal 重构树

- 本质是启发式合并，功能上并没有优越性
- 在运行 Kruskal 算法的过程中，对于两个可以合并的节点 (x, y)，断开其中的连边，并新建一个节点 T，把 T 向 (x,y) 连边作为他们的父亲，同时把 (x,y) 之间的边权当做 T 的点权





Kruskal 重构树

- 这样的结构中，任意两个点路径上边权的最大值为它们的 LCA 的点权，并且每个点能走到其子树中所有点
- 可以发现这个结构也可以解决 Peaks 那样一个点开始只能走边权 <x 这样形式的问题

Solution

- 由于前后都有限定，我们考虑建两颗重构树
- 第一颗按照边权为两个端点编号的最小值构建重构树，重构树每个点的点权 x 表示不经过边权超过 x 的边能到达的所有点；
- 第二颗则按照边权为两个端点最大值来构建重构树，重构树上每个点的点权 x 表示不经过边权不超过 x 的边能到达的所有点。
- 这样我们会在两颗重构树上找到两个点，现在问题就转化成了：这两个点构成的子树中，有没有公共点（因为有公共点 x 的话就肯定存在一条合法路径为 `si->x->ti`）

Solution

- 于是问题转换为：
- 对于两个排列 a, b，每次询问排列 a 在 [l1, r1] 区间内，排列 b 在 [l2, r2] 区间内有无公共元素
- 这个和前面的 CF1093E Intersection of Permutations 类似，只是这里不带修改
- 于是可以二维数点，强制在线所以采用可持久化线段树维护
- O((n+m)logn)

某经典问题

- 给一棵 n 个节点的树，每个点有点权
- m 次查询：
- 给出两条点个数相等的树上简单路径，他们按照点权 sort 之后有哪些位置不相同，要输出每个不同的位置，询问独立，保证输出量 1e6

Solution

- 使用可持久化值域线段树维护链 hash 值，这样每次相当于找出第一个出现次数不同的值
- 每次查询的时候在可持久化 Trie 上面二分这个，可以 O(logn) 的时间内找出一个不同的位置
- 然后暴力一个一个找下去就可以了
- 总复杂度 O(nlogn + tlogn)

Luogu P3302 [SDOI2013] 森林

- 给一个森林，每个点有点权
- 1.link x y，如果 x 和 y 已经连通无视这个操作
- 2.find x y，查询 x 到 y 的链 kth
- 保证时刻是一个森林
- 强制在线

Solution

- 由于这个问题只加边，所以可以考虑一下均摊的方法
- 只加边，维护连通块信息该怎么做呢?
- 启发式合并
- 于是这一题我们可以考虑用启发式合并来处理

Solution

- Link x y 的时候，从 size 小的那个连通块那里开始 DFS，维护树上的可持久化 Trie，相当于把小的连通块的父亲设为大的连通块
- 同时需要维护个倍增表或者 LCT 啥的
- 查询的时候就是普通的链 kth

- O(nlog^2n + mlogn)

Thanks for listening