



Complex Data Structure

By Owaski

Preface

- 这节课主要就是讲讲近年来提出或者被发掘出来的一些复(毒)杂(瘤)数据结构以及它们的应用。
- 先列个目录：
 - 1.吉司机线段树
 - 2.一些支持末尾插入删除的区间信息维护方法
 - 3.可持久化数据结构
 - 4.静态仙人掌
 - 5.动态树
- ~~这是一个普及向的PPT~~
- ~~并不知道能不能讲完~~



1. 吉司机线段树

- 如果你要问我吉司机是谁？我会告诉你无可奉告。



- 反正你知道他很神就对了
- 这个数据结构因为是他搞出来的所以就用他命名了

1.1 区间最值问题

例题 I Gorgeous Sequence

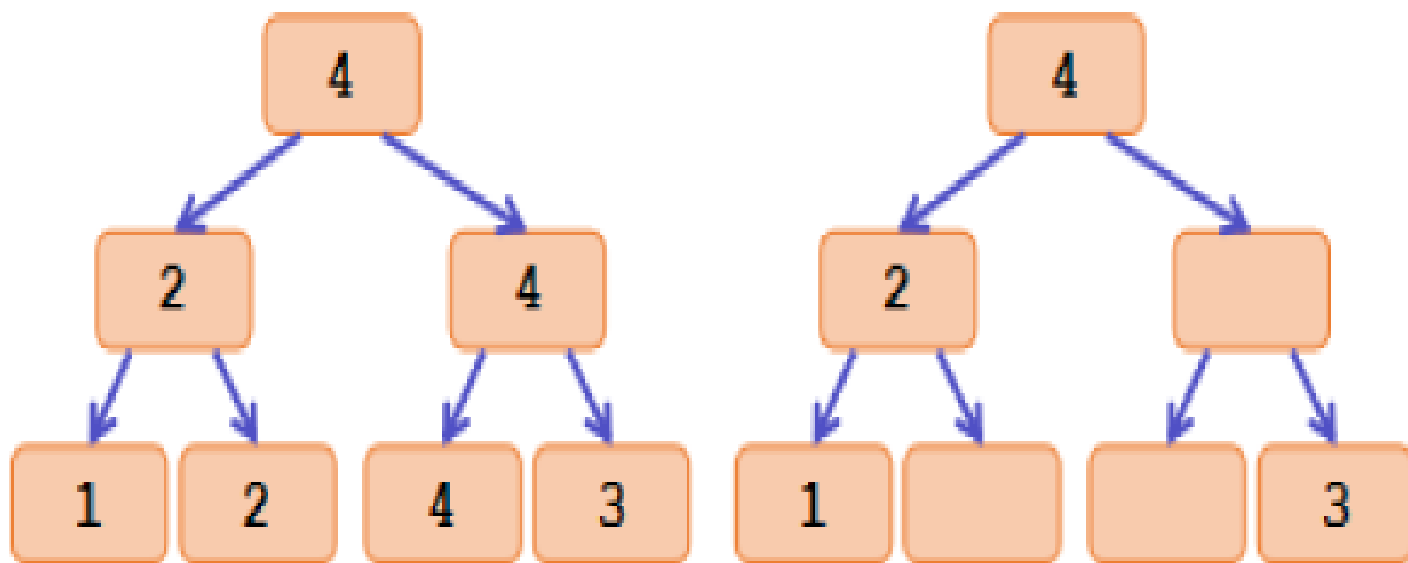
- 给定一个长度为 n 的序列 A ，接下来有 m 次操作，一共3种操作
- a. 给出 l, r, x ，对于 $i \in [l, r]$ ， $A_i = \min(A_i, x)$
- b. 给出 l, r ，对于 $i \in [l, r]$ ，询问 A_i 的最大值
- c. 给出 l, r ，对于 $i \in [l, r]$ ，询问 $\sum_{i=l}^r A_i$
- $n, m \leq 10^6$

例题 I

- 传统的lazy标记并不能够解决问题~~不信你可以试试~~
- 其实这题暴力就好了，当然还是需要建一棵线段树的
- 对每个区间维护区间最大值 ma ，最大值个数 t ，次大值 se ，以及区间和 sum ，还有一个标记 tag
- 对于一个修改操作 l, r, x
- 1. 当 $ma \leq x$ ，不需要做任何操作
- 2. 当 $se < x < ma$ ，显然只会影响到 ma 的值，修改了 ma, sum 后，打个标记退出
- 3. 当 $x \leq se$ ，没办法直接更新这个节点的信息，递归考虑两棵子树
- 考虑下怎么证明复杂度

例题 I

- 将最大值看成标记，那么线段树每个节点都有一个标记，然后将标记值等于父亲标记值的标记删掉，那么可以观察到 se 就是子树标记的最大值。



例题 I

- 定义标记类的概念：
 - 1.同一次取 \min 操作产生的标记属于同一类；
 - 2.同一个标记下传产生的新标记属于同一类；
 - 3.其他的标记都不属于同一类。
- 定义标记类的权值为该标记类所在的线段树节点加上根节点所构成的虚树大小，定义势能函数 $\Phi(x)$ 表示标记类的权值和。
- 那么我们每次定位区间，分裂出的子区间构成的虚树大小是 $O(\log n)$ 的，那么 $\Phi(x)$ 增加了 $O(\log n)$ 。
- 考虑只有当子树中存在标记 $\geq x$ 才会向下暴力 DFS ，回收标记的过程可以看成将该标记类虚树的一棵子树删掉，代价和构成该虚树是一样的。
- 这样的话打标记对 $\Phi(x)$ 的贡献是 $O(m \log n)$ 的，回收的代价也是 $O(m \log n)$ 的，所以整个算法的复杂度也是 $O(m \log n)$ 的。

例题2 Picks loves segment tree

- 给定一个长度为 n 的序列 A ，接下来有 m 次操作，一共3种操作
- a. 给出 l, r, x ，对于 $i \in [l, r]$ ， $A_i = \min(A_i, x)$
- b. 给出 l, r, x ，对于 $i \in [l, r]$ ， $A_i = A_i + x$
- c. 给出 l, r ，对于 $i \in [l, r]$ ，询问 $\sum_{i=l}^r A_i$
- $n, m \leq 3 \times 10^5$

例题2

- 类似上一题的做法，但是复杂度暂时只能证明是 $O(m \log^2 n)$ 。
- $O(m \log^2 n)$ 的证明类似于上题，这不过这题的加减操作会将标记分裂，所以标记类的总数上界从 $O(m)$ 变成了 $O(m \log n)$ 。
- 然而实践检验得到的效率依旧接近 $O(m \log n)$ ，所以猜测标记类的总数依旧是线性的，有兴趣的可以自己探索一下。

小结

- 通过上面的例子可以发现，我们其实是对最值进行加减操作。
- 以最大值为例，我们可以将最大值、其他值分开维护，下传标记的时候传向最大值较大的那棵子树中(相同的时候同时下传)。
- 这样我们得到了一种用 $O(\log n)$ 的代价将区间最值转化成区间加减操作的方法。

1.2 历史最值问题

1.2.1 lazy标记可以解决的问题

- 历史最大值
- 历史最小值
- 历史版本和

例题3 CPU监控

- 给出一个长度为 n 的数列 A ，同时定义一个辅助数组 B ， B 开始与 A 完全相同。接下来进行了 m 次操作，操作有4种类型：
- 1.对于所有的 $i \in [l, r]$ ， $A_i = x$
- 2.对于所有的 $i \in [l, r]$ ， $A_i = A_i + x$
- 3.对于所有的 $i \in [l, r]$ ，询问 A_i 的最大值
- 4.对于所有的 $i \in [l, r]$ ，询问 B_i 的最大值
- 在每一次操作后，我们都进行一次更新，让 $B_i = \max(B_i, A_i)$
- $n, m \leq 10^5$

例题3

- 区间加减我们用 add 标记足矣，就是最后一个操作麻烦一点。
- 我们对每个节点记一个历史最大标记 pre ， pre 表示上次把这个节点的标记下传到当前时刻的这段时间中， add 的最大值。
- 下传的时候标记是很容易合并的：
- $add_{left} = add_{left} + add_x$
- $pre_{left} = \max(pre_{left}, pre_x + add_{left})$
- 更新历史最大值只需要拿当前区间最大值+ pre 即可。
- 对于区间覆盖，类似考虑。
- 时间复杂度 $O(m \log n)$ 。

例题4 V

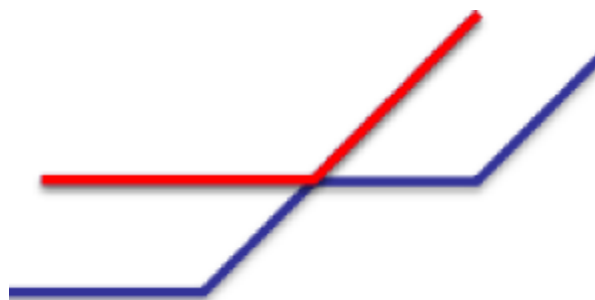
- 给出一个长度为 n 的数列 A ，同时定义一个辅助数组 B ， B 开始与 A 完全相同。接下来进行了 m 次操作，操作有5种类型：
- 1.对于所有的 $i \in [l, r]$ ， $A_i = \max(A_i - x, 0)$
- 2.对于所有的 $i \in [l, r]$ ， $A_i = A_i + x$
- 3.对于所有的 $i \in [l, r]$ ， $A_i = x$
- 4.给出一个 i ，询问 A_i 的值
- 5.给出一个 i ，询问 B_i 的值
- 在每一次操作后，我们都进行一次更新，让 $B_i = \max(B_i, A_i)$
- $n, m \leq 5 \times 10^5$

例题4

- 先定义一种标记 (a, b) ，表示将数先 $+a$ 再对 b 取 \max 。那么前三种操作分别是 $(-x, 0), (x, -INF), (-INF, x)$ 。
- 考虑合并两个标记 $(a, b), (c, d) \rightarrow (a + c, \max(b + c, d))$
- 现在解决最后一个询问，我们依旧记历史最大标记。
- 考虑到现在的标记 (a, b) 并不是单纯的数，但是我们可以将其看成关于 A_i 的分段函数
- $$(a, b) = \begin{cases} b, & A_i \leq b - a \\ A_i + a, & A_i > b - a \end{cases}$$

例题4

- 这个函数图像是先斜率为0，然后斜率为1的折线，很显然如果要取历史最大的话，我们可以取其上轮廓。



- 之后就和上道题没啥区别了，复杂度 $O(m \log n)$

小结

- 以上是可以 lazy 标记解决的历史最值问题，大致思路都是记录一个历史最值标记表示从上次下传到当前时刻的最值。

1.2.2 lazy标记无法处理的问题

1.2.2.1 无区间最值

- 区间加减
- +
- 区间历史最大值之和
 - 设一个 $C_i = B_i - A_i$, 如果 $A_i + x > B_i$, $C_i = 0$, 否则 $C_i = C_i - x$, 换句话说 $C_i = \max(C_i - x, 0)$, $O(m \log^2 n)$
- 区间历史最小值之和
 - 设一个 $C_i = A_i - B_i$, 如果 $A_i + x < B_i$, $C_i = 0$, 否则 $C_i = C_i + x$, 换句话说 $C_i = \max(C_i + x, 0)$, $O(m \log^2 n)$
- 区间历史版本和之和
 - 设一个 $C_i = B_i - T \cdot A_i$, 那么 $newC_i = (B_i + A_i + x) - (T + 1) \cdot (A_i + x) = C_i - T \cdot x$, $O(m \log n)$

1.2.2.2 有区间最值

- 之前讨论过将区间最值转化成区间加减法的做法，将其与无区间最值的做法嵌套起来即可，当然时间复杂度也是嵌套的。

总结

- 上述讨论的关于线段树的问题都是之前OI不常见的，但非常有潜力的问题——区间最值和历史最值问题，这里还只是介绍了一下，这类问题可以多研究一下。~~然后出题去坑别人~~



2. 一些支持末尾插入删除的 区间信息维护方法

- 这玩意儿好像是去年lzz提出来的，至于你问我lzz是谁，我只能回你一句无可奉告。
- (然而并没有找到照片) 
- 反正你知道他很神就对了
- 他去年出了一道题，拿出来讲讲，提升一下姿势水平

例题5 Unknown

- 有一个元素为向量的序列 S ，下标从1开始，初始时 S 为空，现在你需要支持3个操作：
- 1.在 S 的末尾添加一个元素 (x, y)
- 2.删除 S 的末尾元素
- 3.询问下标在 $[l, r]$ 区间内的元素中， $(x, y) \times S_i$ 的最大值
- \times 表示叉积， $(x_1, y_1) \times (x_2, y_2) = x_1y_2 - x_2y_1$

测试点编号	m 的规模	其他
1	$m \leq 1000$	无
2		
3		
4	$m \leq 80000$	无2操作, 3操作的询问为全部区间
5		所有2操作在1操作的后面, 3操作的询问为全部区间
6		所有3操作都在1操作和2操作后面
7	$m \leq 300000$	对于所有3操作有 $l = 1$, 内存限制为128M
8		无
9		内存限制为128M
10		内存限制为64M

对于全部数据, 令 n 为任意时刻序列长度, $n \leq 300000; m \leq 500000$ 。

1操作个数不超过300000, 且输入的 x, y 满足 $-10^9 \leq x \leq 10^9; 1 \leq y \leq 10^9$ 。


3操作个数不超过300000, 且输入的 x, y 满足 $1 \leq x \leq 10^9; -10^9 \leq y \leq 10^9$ 。

3操作中, l, r 满足 $1 \leq l \leq r \leq n$ 。

叉积的性质

- $\overrightarrow{OA} \times \overrightarrow{OB} = |\overrightarrow{OA}| |\overrightarrow{OB}| \sin\theta$, θ 表示 \overrightarrow{OA} 和 \overrightarrow{OB} 的逆时针角度
- 可以看出 $|\overrightarrow{OB}| \sin\theta$ 其实就是 B 到直线 OA 的距离, 因此我们只需要维护 \overrightarrow{OB} 构成的上凸壳即可, 也就是 S 集合构成的上凸壳。

算法I 无脑分块(在线)

- 假设将序列分成 S 块，那么块数为 $\frac{n}{S}$ ，插入和删除都要重建最后一个块的凸壳，复杂度为 $O\left(\frac{n}{S}\right)$
- 询问的时候对每个块内的凸壳做二分，复杂度为 $O(S \log n)$
- $\frac{n}{S} = S \log n$ ，显然取 $S = \sqrt{\frac{n}{\log n}}$ 的时候最优
- 时间复杂度为 $O(n\sqrt{n \log n})$
- 太慢了有木有 

算法2 点分治 I (离线)

- 考虑离线，插入和删除可以构建出一棵操作树出来，那么每一次询问一个区间就相当于是在树上一条路径，那么考虑点分治，处理经过重心 G 的所有询问 (x, y) ，可以拆成 (x, G) 和 (G, y) 所以问题变成了维护一条路径上的凸壳。
- 由于要支持插入和撤销上次插入两种操作，我们可以采用平衡树维护凸壳，插入直接二分插入，撤销直接删掉新的点，把原来的点加上。
- 对于一棵子树是 $O(n \log n)$ 的，总复杂度是 $O(n \log^2 n)$
- 由于点分治的复杂度是 $O(n)$ 的，是可以AC的。

算法3 点分治2(离线)

- 考虑因为这题的路径都是从下往上的，对于一个点分治结构，设重心为 G ，深度最浅的点为 R ，考虑 $R \rightarrow G$ 这条路径，每个询问都必定在上面是一个后缀，因此我们从 $G \rightarrow R$ 加点动态维护凸壳即可。
- 时空复杂度都和上个做法一样。

算法4 点分治3(离线)

- 还是类似上个做法，考虑 $G \rightarrow R$ ，询问必定是后缀，那么我们把询问挂在路径上，对这条路径做分治凸包，因为排序可以在归并的时候解决所以复杂度是 $O(n \log n)$ 。
- 再加上点分治就和上题的复杂度是一样的了。

算法5 线段树(离线)

- 我们考虑用树链剖分将链剖成 $O(\log n)$ 段 dfs 序区间，然后问题变成了询问区间的凸壳。
- 用线段树维护区间的凸壳
- 因为一个询问被拆成了 $O(\log n)$ 个，被拆出来的子询问在线段树中又要被拆成 $O(\log n)$ 个区间，因此一共有 $O(n \log^2 n)$ 个询问，我们可以将所有询问按极角排序插入到线段树中，然后直接按极角序扫一遍线段树即可。
- 时间 $O(n \log^2 n)$ ，但是空间 $O(n \log^2 n)$ 大了点。

算法6 二进制分组(在线)

- 如果没有删除的话，二进制分组是长这样的。
- 每次加入一个元素到末尾单独成为一块，如果末尾两个块大小相同，那么合并它们，一直合并到不能合并为止。
- 考虑一个元素所在的块每次合并大小会翻倍，因此一个元素贡献的复杂度是 $O(\log n)$ ，所以构建的复杂度是 $O(n \log n)$ 的。
- 考虑每次查询，将查询最多分成 $O(\log n)$ 个块，再加上二分的复杂度就是 $O(n \log^2 n)$ 。
- 如果有删除的话，单次重构的复杂度可能到 $O(n)$ ，总复杂度就是 $O(n^2)$ 了，所以我们需要改进这个做法。

算法6 二进制分组(在线)

- 考虑换一种建二进制分组的方式。
- 建立 $t + 1$ 层结构, t 为满足 $2^t \geq n$ 的最小 t 。
- 第 i 层的组的大小为 2^i , 有 $\left\lfloor \frac{n}{2^i} \right\rfloor$ 组, 长得很像一棵线段树。
- 插入的话差别不大, 从0层开始往上合并。
- 考虑删除, 引入替罪羊思想, 对于要重构的组, 我们打个标记, 并不立马重构, 但也不能一直不重构, 不然询问的时候访问到的节点数会爆炸。
- 我们约定: 每一层至多允许最后一个组是有标记的。
- 这样的话, 当第 i 层需要增加一个组的时候才会将原来的最后一组重构, 当这个组被标记完了, 那么就直接变成空的即可。
- 考虑怎么算这个的复杂度。

算法6 二进制分组(在线)

- 很显然我们只需要证明重构的复杂度即可。
- 对于第 i 层，如果最后一个组要重构，那么至少已经加入了 2^i 个元素也就是形成了一个新的块，重构的复杂度也是 $O(2^i)$ 的，因此重构的复杂度和加入元素的复杂度是一样的，所以维护这个结构的复杂度就是 $O(n \log n)$ 的。
- 对于询问，将区间 $[l, r]$ 在结构中拆成子区间，有三种情况：
 - 1.该区间没被打标记，直接查询
 - 2.该区间打了标记，但左右孩子都没被打标记，直接查询孩子
 - 3.该区间打了标记，右孩子也有标记，先查询左孩子，然后递归右孩子，因为打标记的区间个数是 $O(\log n)$ 的，所以三种情况的复杂度合起来还是 $O(n \log n)$ 的，加上二分的复杂度，总时间复杂度就是 $O(n \log^2 n)$ ，但是空间复杂度是 $O(n \log n)$ 的。

总结

- 这道题总归是有几个不常见的思路，离线就操作树点分治，在线二进制分组等等，对于末尾插入删除的这类题，这几种方法几乎都能适用，而且各有长处，可以根据需求进行取舍。



3.可持久化数据结构

定义

- 可持久化数据结构顾名思义就是一种保存了每次修改的版本的
数据结构，可持久化分为部分可持久化以及完全可持久化。
- 部分可持久化表示可以访问之前的版本，但只能从当前最新的
状态更新出下一个版本的数据结构。
- 完全可持久化表示可以访问之前的版本，而且能从之前任意一
个版本更新出下一个版本的数据结构。

例子

- 支持可持久化的数据结构有许多：
- 链表、树、栈、线段树、平衡树、*trie*树等等

构建

- 链表和树本身就支持可持久化
- 栈可以看成一棵操作树来进行可持久化
- 那我们先花一点时间讲一下线段树和平衡树的可持久化
- ~~虽然你们大部分都知道~~



可持久化线段树

- 考虑到每次对线段树的修改，涉及的结点个数都是 $O(\log n)$ 的
- 那我们每次只需要新建 $O(\log n)$ 个结点即可，新建结点的左右孩子根据左右孩子是否被修改考虑连向原来的结点还是新的结点，看下单点插入的可持久化线段树伪代码。

```
node insert(node x, position, value) {
    node ret = newnode
    if position in x->lchild
        ret->rchild = x->rchild
        ret->lchild = insert(x->lchild, position, value)
    else
        ret->lchild = x->lchild
        ret->rchild = insert(x->rchild, position, value)
    return ret
}
```

可持久化平衡树

- 一般采用可持久化非旋转 $treap$ ，先说一下啥是非旋转 $treap$ 。
- 我们只要支持合并两棵 $treap$ 和从某个点分裂一棵 $treap$ 两种操作即可实现一般的平衡树操作。
- 考虑怎么合并两棵 $treap$ ，考虑 $merge(u, v)$ ，其中 u 的 $value$ 整体小于 v 的 $value$ ，如果 $u \rightarrow key > v \rightarrow key$ ，那么将 v 作为根， $v \rightarrow lc = merge(v \rightarrow lc, u)$ ， $v \rightarrow rc$ 不变，反之亦然，最后将 v 返回即可。
- 根据 $treap$ 的期望深度可以知道单次复杂度是 $O(\log n)$ 的。
- 分裂的话方法类似，就不再赘述了。

可持久化平衡树

- 至于可持久化，只要每次返回的结点都是新建立的结点即可。

```
node merge(node u, node v) {  
    if u is null  
        return v  
    if v is null  
        return u  
    node ret = newnode  
    if u->key < v->key  
        ret->key = u->key  
        ret->value = u->value  
        ret->rc = merge(u->rc, v)  
        ret->lc = u->lc  
    else  
        ret->key = v->key  
        ret->value = v->value  
        ret->lc = merge(v->lc, u)  
        ret->rc = v->rc  
    return ret  
}
```

为啥不用splay呢？

- 因为splay的复杂度会出问题，splay在不持久化的时候是用势能分析复杂度的，也就是说单次操作的代价可能会很高，但是在可持久化后，我们可以不断重复代价高的那次操作以至于达到 $O(n^2)$ 复杂度。
- 而treap由于期望深度小而且形态固定因此普遍被使用。

例题6 BZOJ3674

- n 个元素 m 个操作，操作分为三种：
- 1 $a\ b$ 将 a, b 所在集合合并
- 2 k 回到 k 次操作之后的状态(查询算操作)
- 3 $a\ b$ 询问 a, b 是否属于同一集合
- 强制在线
- $n, m \leq 2 \times 10^5$

例题6

- 可持久化并查集
- 反正并查集可以用数组实现，那就用可持久化线段树来维护数组，注意不要路径压缩，但是要按秩合并，复杂度 $O(n \log^2 n)$
- 如果路径压缩会破坏之前的信息从而不好维护

例题7 区间第 k 大

- 给定一个长为 n 的序列 a ，有 m 次操作。
- 每次操作询问区间 $[l, r]$ 中第 k 大的元素的值。
- $n, m \leq 5 \times 10^5$

例题7

- 可持久化线段树的经典应用。
- 核心思想就是考虑怎么提出区间 $[l, r]$ 的权值线段树。
- 我们考虑从 $1 \sim n$ 逐次将 a_i 加入到线段树中，并且是可持久化的加入，也就是说保存了每次加入后的版本，加入了前 i 个元素的线段树记为 T_i 。
- 因为权值线段树是可加减的，因此区间 $[l, r]$ 的权值线段树就是 $T_r - T_{l-1}$ ，在这棵线段树中查询第 k 大的值即可。
- 时空复杂度都是 $O(n \log n)$ 。

例题7 拓展

- 如果支持单点加减怎么办呢?
- 有三种方法:
 1. 树状数组套可持久化线段树
 2. 权值线段树套线段树
 3. 权值线段树套平衡树
- 时空复杂度分别是:
 1. $O(n \log^2 n)$, $O(n \log^2 n)$ (貌似可以做到 $O(n \log n)$)
 2. $O(n \log^2 n)$, $O(n \log^2 n)$
 3. $O(n \log^2 n)$, $O(n \log n)$

例题8 谈笑风生

- 给出一棵 n 个点的树，以及 Q 个询问，每个询问形式如下：
- 给定 p, k ，问有多少个三元组 (p, x, y) ，满足
- p 和 x 都是 y 的祖先
- p 和 x 的距离小于等于 k
- $n, Q \leq 3 \times 10^5$

例题8

- 考虑 x 要么是 p 的祖先，要么是 p 的子树中的结点。
- 考虑一个 y 对应的 x 的数量(根结点的深度为0):
$$\min(\text{depth}_y - \text{depth}_p - 1, k) + \min(\text{depth}_p, k)$$
- 后面一半对于所有 y 都是一样的，考虑前面一半。
- 很显然我们可以把深度 $\leq \text{depth}_p + k$ 和 $> \text{depth}_p + k$ 的 y 分开统计，前者的贡献就是 $\text{depth}_y - \text{depth}_p - 1$ ，后者的贡献就是 k ，因此我们只需要分别计算这两种 y 的答案即可。
- 我们现在要统计的就是在一棵子树中深度 \leq 某个值得点的个数以及深度和，考虑子树在 dfs 序上是一个区间，可以转化成求一个区间中深度 \leq 某个值的点的个数。
- 很显然我们直接按 dfs 序，以深度为比较键值建立可持久化线段树即可，对于一个区间 $[l, r]$ ，只需要拿 r 时的权值线段树减去 $l - 1$ 时的权值线段树即可得出这个区间的线段树。

例题9 basic persistent treap

- 初始给定一个长为 n 的序列 a ，有 m 次操作，分别有4种操作
- 1. t, l, r, k 将 t 时刻的序列区间 $[l, r]$ 加上 k 并将其作为新的序列
- 2. t, l, r 将 t 时刻的序列区间 $[l, r]$ 翻转并将其作为新的序列
- 3. t, l, r 询问 t 时刻序列区间 $[l, r]$ 的最大值
- 4. t, l, r 询问 t 时刻序列区间 $[l, r]$ 的最大值的前驱的值
- $n, m \leq 10^5$

例题9

- 我们已经知道了可持久化*treap*的*merge*, *split*操作如何进行。
- 那么对于每个区间操作，我们先将其*split*出来，然后打个*lazy*标记，再将其合并回去并形成一棵新的*treap*。
- 时空复杂度都是 $O(n \log n)$ ，但是常数较大。

例题 10 oath

- 交互题，维护一种数据结构，支持从末尾插入删除，要求可持久化，并且支持区间“求和”，这里的“求和”是指定义的一种满足交换结合律的二元运算 $F(x, y)$ 的求和。要求求和的过程中 F 的调用次数不超过一次。
- 操作次数 $\leq 3 \times 10^5$ ，交互库会根据你的程序运行情况生成之后的询问。

例题 10

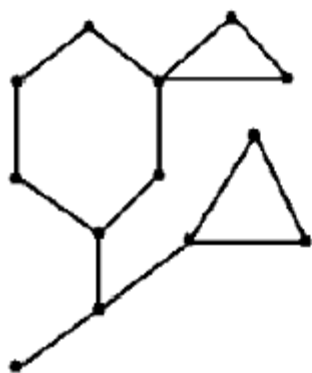
- 考虑怎样让询问只要调用一次 F ，构建出操作树，区间 $[l, r]$ 相当于某个结点 p 到根的链的一个后缀。
- 考虑 p 的信息，如果我们保存 p 到根的链上的所有后缀信息，每加入一个点，需要打 $lazy$ 标记，询问的时候因为要下传 $lazy$ 标记，因此不能保证询问只调用一次 F 。
- 考虑用可持久化 $treap$ 存储信息，并对每个结点保存两个信息
- 假设一棵子树的结点的顺序是 $a_1, a_2, \dots, a_{root}, \dots, a_{size}$ ，对应的值为 v 。
- 1. $pre_0 = v_{root}, pre_i = F(a_{root-i}, pre_{i-1})$
- 2. $suf = F(a_{root+1}, \dots, a_{size})$
- 那么对于每次询问，我们只要要在 p 的 $treap$ 中，找到一个包含了 $[l, r]$ 区间的结点，然后做 $F(pre_{pos-l}, suf)$ 即可。
- 因为 $treap$ 子树大小是期望 $O(\log n)$ 的，所以时空复杂度都是 $O(n \log n)$ 的



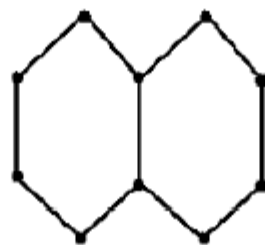
4.静态仙人掌

定义

- 如果一个无向连通图的任意一条边属于最多一个简单环，且没有自环，我们就称之为仙人掌。



仙人掌 ✓



不是仙人掌 ✗
(中间那条边在两个环上)



不是仙人掌 ✗
(不是连通图)

边数

- 考虑仙人掌的一棵 n 个结点的生成树，有 $n - 1$ 条边，剩下的边皆为非树边，而且每条非树边对应了生成树上的一条路径并形成了一个简单环。
- 定义要求每条边最多属于一个简单环，所以每条非树边对应的路径都不相交，最多有 $n - 1$ 条这样的路径，所以 n 个结点的仙人掌，最少有 $n - 1$ 条边，最多有 $2n - 2$ 条边。

父亲

- 定义结点的父亲和环的父亲
- 如果点 p 到根结点之间所有简单路径的第一条边相同，那么和树的情况是一样的，父亲为第一条边对应的另一个点，否则父亲为第一条边所在的简单环。
- 环的父亲为环上距离根最近的一点。

儿子

- 定义结点上的儿子和环的儿子。
- 儿子的定义和父亲的定义是相反的，可以类比。

父亲结点和母亲结点

- 对于一个在环上的结点，我们定义它的**父亲结点**和**母亲结点**分别为环上与它相邻的两个结点(请注意区分**父亲**和**父亲结点**)。

遍历

- 假设当前在结点 x ，接下来的点是 y 。
- 如果 y 还没访问过，那么访问 y
- 如果 y 访问过，并且在 x 之前访问，那么我们找到了一个简单环
- 如果 y 访问过，并且在 x 之后访问，这个环已经被考虑过了，因此可以忽略

仙人掌 DP

- 和树形 DP 类似，需要分开考虑树边和环的 DP 。

例题 II 经典问题

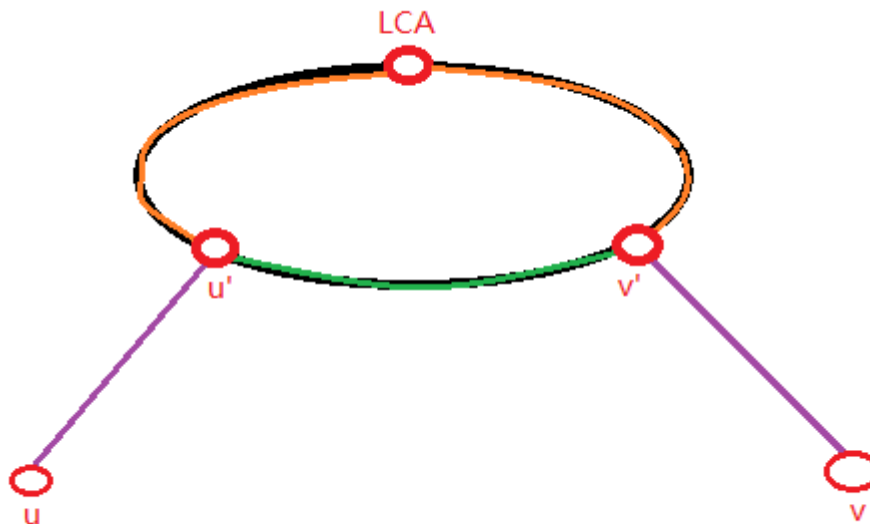
- 给一棵 n 个结点的仙人掌，每条边有个长度，求1号结点到每个结点的最短路径长度。
- $n \leq 5 \times 10^6$

例题 II

- dfs 一遍搞出结构，然后从1号结点开始 DP 。
- 考虑当前走到结点 x ，考虑 x 的每个儿子，如果儿子是结点，那么可以直接转移，否则儿子是一个环，两条路径取短的那条去更新即可。
- 复杂度 $O(n)$ 。

例题II 拓展

- 假如给你 m 次询问，每次询问一对点的最短路径怎么做？
- 类比树的做法，求出 LCA 即可，但是仙人掌上的 LCA 可以在环上，如图：



例题 II 拓展

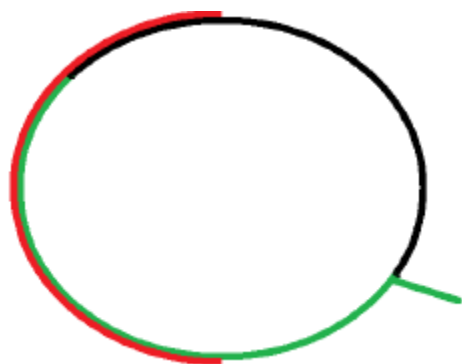
- 考虑将 u, v 先倍增到 u', v' ，环上的路径要么是橙色路径要么是绿色路径，求个最小值即可。
- 由于有倍增，复杂度是 $O(n \log n)$ 。

例题12 另一个经典问题

- 给一棵 n 个结点的仙人掌，每条边有个长度，求直径。
- 仙人掌直径的定义是最大的两点间最短路径
- $n \leq 5 \times 10^6$

例题 12

- 树的直径可以两遍 dfs ，但是仙人掌这样搞有可能会挂。



- 考虑一个 $2k$ 个结点的环再加上一条边，边权都是1，直径为 $k + 1$ ，但是只有当第一遍 dfs 的起始点为某些点的时候才能得出这个结果。

例题12

- 考虑树形 DP 的时候我们需要知道以 x 为根的子树的最大深度。
- 类比树形 DP ，我们可以对每个结点 x 求出子仙人掌 x 的最大深度，接着考虑怎么计算答案。
- 对于每个结点，我们用不同儿子的最大值和次大值。
- 对于每个环，需要考虑每对结点的最大深度和这对结点的最短距离。
- 不妨枚举其中一个结点 p ，那么最短路是顺时针的一条路径，这条路径可以看成一个区间，随着 p 的移动而移动，这样就变成了左右端点都在单调移动的 RMQ 问题，可以用单调队列解决
- 当然肯定不止这种方法，但复杂度都可以做到 $O(n)$ 。

例题13 mx的仙人掌

- 给一棵 n 个点的仙人掌和 q 个询问，每个询问给定一个点集，要求输出这个点集中距离最远的点对的距离。两个结点之间的距离为它们的最短路径长度。
- $n, q \leq 3 \times 10^5$, 点集大小之和 $\leq 3 \times 10^5$
- 时间限制5s，空间限制512MB

例题 13

- 先考虑树的情况， $f[x][i]$ 表示子树 x 中第 i 个询问的点集中到点 x 的最大距离，然后树形 dp 的时候合并这些数组，同时更新答案
- 合并可以采用启发式合并， $f[x]$ 的信息可以用一个哈希表存起来(`unordered_map`好评)，这样复杂度就是 $O(n \log n)$ 了。

例题 13

- 对于树边，和树上一样的处理。
- 对于每个环，先将环的每个儿子的 dp 数组处理出来，然后用每对 dp 数组的交集来更新答案。
- 考虑根据到环的父亲距离将环分成两部分，分别合并两部分的 dp 数组，合并时更新两部分答案，然后再求两个部分的 dp 数组的交集来更新答案。
- 然后撤销所有合并操作，最后将所有儿子的 dp 数组合并。
- 复杂度 $O(n \log n)$ 。

例题 13

- 感觉上虚仙人掌是能建出来的，这样就可以在线啦，复杂度应该还是 $O(n \log n)$ 。

仙人掌点分治

- 每次找一个结点(我们称它为重心), 使删掉这个结点和这个结点所在的所有环上的边之后, 最大的连通块大小最小, 然后统计环上的信息, 再递归分治剩下的联通块。
- 考虑当前连通块大小为 N , 如果选择了结点 x , 并且存在一个结点 y , 使以 x 为根, 子仙人掌 y 的大小大于 $\frac{N}{2}$, 那么选择 y 一定比选择 x 要优。所以每次分治后, 最大的连通块大小至少减少了一半。所以分治的层数是 $O(\log n)$ 的。

例题14 跳蚤国王下江南

- 给一棵仙人掌，求对于 $l = 1, 2, \dots, n$ ，从1号点出发经过 l 条边的简单路径条数，答案模998244353。
- $n \leq 10^5$

例题 14

- 考虑暴力 dp
- 记 $f[x][l]$ 表示从结点 x 开始，只能向子仙人掌的方向走，长度为 l 的路径条数。
- 如果 x 的儿子 y 是结点，那么 $f[x][l] += f[y][l - 1]$
- 如果 x 的儿子是一个环，考虑环上非 x 的点 y ，那么 $f[x][l] += f[y][l - d_1] + f[y][l - d_2]$ ，其中 d_1, d_2 分别是 $x \rightarrow y$ 的两条路径长度。
- 这个的复杂度是 $O(n^2)$ 的，需要优化。

例题 14

- 可以将 $f[x]$ 看成一个多项式，那么更新可以看成：
- 1. $f[x] += f[y] \cdot x$
- 2. $f[x] += f[y] \cdot (x^{d_1} + x^{d_2})$
- 考虑仙人掌分治，设当前根结点为 R ，重心为 G ，我们先递归不包含 R 的子仙人掌，求出一个多项式 g ，然后递归 R 所在的子仙人掌，保存每个分治结构重心到根的多项式，逆着乘起来就可以得到 $R \rightarrow G$ 的多项式 f 了，用 $f \cdot g$ 去更新 R 的多项式即可。
- 总复杂度是 $O(n \log^2 n)$ 的。

总结

- 该节主要讲了仙人掌上的 dp 以及分治如何应用，动态仙人掌在 OI 中考察的几率不大，因此对于静态的仙人掌需要掌握这些基础知识。



5.动态树问题

动态树问题

- 动态树问题是指修改树上的信息，同时询问树上信息的一类问题，下面先来介绍一个有力的工具。

Link/cut tree

- 一种动态维护树上信息的结构，简称 LCT ，支持在 $O(\log n)$ 时间内完成下面的操作：
 1. $cut(u, v)$ 删除边 (u, v)
 2. $link(u, v)$ 加入边 (u, v)
 3. $findroot(x)$ 找到 x 所在树的根结点
 4. $beroot(x)$ 将 x 变成树根
- ...还有很多操作就不一一列举了

Link/cut tree

- 称一个点被访问过，如果刚刚执行了对这个点的 $access$ 操作
- 如果结点 v 的子树中，最后被访问的结点在子树 w 中，这里 w 是 v 的儿子，那么就称 w 是 v 的Preferred Child。
- 如果最后被访问过的结点就是 v 本身，那么它没有Preferred Child。
- 每个点到它的Preferred Child的边称作Preferred Edge，也可称为实边。
- 由Preferred Edge连接成的不可再延伸的路径称为Preferred Path。

Link/cut tree

- 这样，整棵树就被划分成了若干条Preferred Path。
- 对每条Preferred Path，将这条路上的点的深度作为关键字，用一棵平衡树来维护它。
- 在这棵平衡树中，每个点的左子树中的点，都在Preferred Path中这个点的上方；右子树中的点，都在Preferred Path中这个点的下方。
- 需要注意的是，这种平衡树必须支持分离与合并。这里我们选择Splay Tree作为这个平衡树的数据结构。我们把这棵平衡树称为一棵Auxiliary Tree。

Link/cut tree

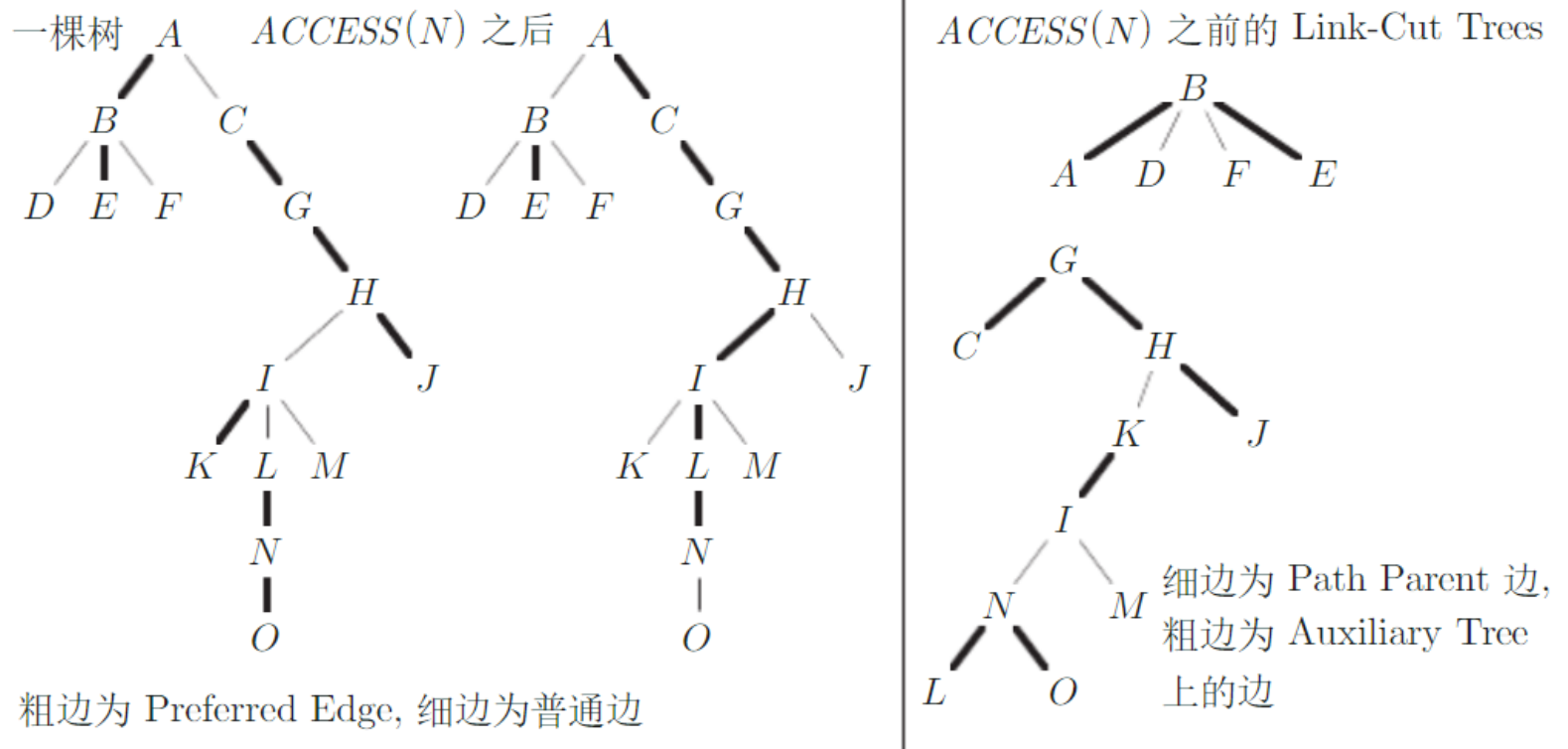
- 知道了树 T 分解成的这若干条 Preferred Path，我们只需要再知道这些路径之间的连接关系，就可以表示出这棵树 T 。
- 用 Path Parent 来记录每棵 Auxiliary Tree 对应的 Preferred Path 中的最高点的父亲结点，如果这个 Preferred Path 的最高点就是根结点，那么令这棵 Auxiliary Tree 的 Path Parent 为 *null*，Path Parent 和 Preferred Path 中的最高点的父亲结点之间的边称为虚边。
- *LCT* 就是将要维护的森林中的每棵树 T 表示为若干个 Auxiliary Tree，并通过 Path Parent 将这些 Auxiliary Tree 连接起来的数据结构。

access

- *access*操作 *LCT* 的所有操作的基础。假设调用了过程 *access(v)*，那么从点 *v* 到根结点的路径就成为一条新的 Preferred Path。
- 如果路径上经过的某个结点 *u* 并不是它的父亲 *parent(u)* 的 Preferred Child，那么由于 *parent(u)* 的 Preferred Child 会变为 *u*，原本包含 *parent(u)* 的 Preferred Path 将不再包含结点 *parent(u)* 及其之上的部分。

access

- 下图为 *LCT* 的一个结构示意图，及一次 *access* 操作的对比图



access

- 首先,由于访问了点 v , 那么它的Preferred Child应当消失。
- 先将点 v 旋转到它所属的Auxiliary Tree的根, 如果 v 在 v 所属的Auxiliary Tree中有右儿子(也就是 v 原来的Preferred Child), 那么应该将 v 在 v 所属的Auxiliary Tree中的右子树(对应着它的原来的Preferred Child之下的Preferred Path)从 v 所属的Auxiliary Tree中分离, 并设置这个新的Auxiliary Tree的Path Parent为 v

access

- 然后，如果点 v 所属的Preferred Path并不包含根结点，设它的Path Parent为 u ，那么需要将 u 旋转到 u 所属的Auxiliary Tree的根。
- 并用点 v 所属的Auxiliary Tree替换到点 u 所属的Auxiliary Tree中点 u 的右子树。
- 再将原来点 u 所属的Auxiliary Tree中点 u 的右子树的Path Parent设置为 u 。
- 如此操作，直到到达包含根结点的Preferred Path。
- 可以证明它的复杂度是均摊 $O(\log n)$ 的。
- 可以将Preferred Path上的边称为实边，

边信息与点信息

- 有些题要对点的信息进行维护，而有的要对边的信息做维护，通常有两种方法：
- 1. 对边 (u, v) ，新建结点 x ，将边 (u, v) 改成 $(u, x) + (x, v)$ 两条边，边的信息记在 x 上。
- 2. 另一种方法是同时记录一条链与每个点关联的两条边，在换根的时候维护。

实现

- 1. $findroot(x)$, 对 x 进行 $access$, 找到该 $splay$ 中最左端的点
- 2. $beroot(x)$, 对 x 进行 $access$, 旋转到该 $splay$ 中顶端, 然后打上翻转标记
- 3. $link(u, v)$, 将 u 设为根, 然后将 u 的 Path Parent 设为 v
- 4. $cut(u, v)$, 对边的一个端点进行 $access$, 将 (u, v) 变成虚边, 然后直接删除
- 容易发现, 序列上的区间询问, 很容易通过 LCT 推到树上, 比如要询问链 (u, v) , 先 $beroot(u)$, 然后 $access(v)$, 那么 u, v 所在的 $splay$ 就是链 (u, v) 。

经典问题 I 动态最小生成树

- 强制在线，只有加边操作，动态维护最小生成树。
- 对于加入的一条边 (u, v) ，考虑链 $u \rightarrow v$ 上的最大边 (x, y) ，如果 $w(u, v) < w(x, y)$ ，那么将边 (x, y) 删除，加入 (u, v) ，否则不进行操作。

经典问题2 最小极差路径

- 给定一个带权无向图，求从 $S \rightarrow T$ 的一条路径，树的路径上的最大权值减最小权值最小。
- 将边升序排序，从小到大枚举最大权值，相当于询问在权值小于等于当前最大权值的边构成的图中从 $S \rightarrow T$ 的最大瓶颈路，实时维护一个最大瓶颈树，也就是最大生成树即可。

经典问题3 动态二分图

- 给定一张图，可以加边或者删边，每次求是否是二分图，允许离线。
- 离线搞出每条边的删除时间，维护关于离线时间的最大生成树
- 对于加边，如果形成环，那么删除环上最早删除的边。删除边的時候还要判断这个边与树边是否形成了奇环，如果是那么加入一个集合。
- 删边的时候，如果该边是树边，那么直接删除，否则若在集合中那么从集合中将其删除，如果某个时刻集合为空则是二分图

例15 QTREE

- 给定一棵 n 个结点的带边权的树，有 m 次操作，有2种操作：
- 1. 修改边权
- 2. 询问两点间边权最大值
- $n, m \leq 10^5$

例16 QTREE2

- 给定一棵 n 个结点的带边权的树，有 m 次操作，有2种操作：
- 1. 询问两点间距离
- 2. 询问 $u \rightarrow v$ 路径上第 k 个点的编号
- $n, m \leq 10^5$

例题 I7 QTREE3

- 给定一棵 n 个结点的树，每个点初始都是白色的。
- 有 m 次操作，有2种操作：
 1. 反转一个点的颜色
 2. 询问 $u \rightarrow v$ 路径上的第一个黑点
- $n, m \leq 10^5$

例题 I8 QTREE4

- 给定一棵 n 个结点的树，每个点初始都是白色的。
- 有 m 次操作，有2种操作：
 1. 反转一个点的颜色
 2. 询问整棵树中最远的两白点之间的距离
- $n, m \leq 10^5$

例题 18

- 这题有很多做法，点分治，树剖等，这里讲 LCT 的做法。
- 考虑要维护最远的两白点距离，我们需要知道以 x 为根向下的最远的白点的距离。
- 同时因为要维护子树信息，我们要用两个 $multiset$ ，一个用来求出点 x 向下通过虚边的最远白点，另一个是虚边下面的子树的答案。
- 剩下的就是经典的问题了。
- 复杂度 $O(n \log^2 n)$ 。

例题 19 QTREE5

- 给定一棵 n 个结点的树，每个点初始都是黑色的。
- 有 m 次操作，有2种操作：
 1. 反转一个点的颜色
 2. 询问离 x 最近的白点
- $n, m \leq 10^5$

例题 19

- 这题比上一题要简单，一个 *multiset* 就足够了。

例题20 QTREE6

- 给定一棵 n 个结点的树，每个点初始都是黑色的。
- 有 m 次操作，有2种操作：
 1. 反转一个点的颜色
 2. 询问有多少个点与 u 相连，两个结点 u, v 相连当且仅当 $u \rightarrow v$ 路径上所有点的颜色相同。
- $n, m \leq 10^5$

例题20

- 维护左右端点的颜色，和左右端点相连的点的数量，自己的颜色，子树的两种颜色点的数量即可。

例题2 I QTREE7

- 给定一棵 n 个结点的带点权的树，每个点有初始黑白颜色。
- 有 m 次操作，有3种操作：
 1. 反转一个点的颜色
 2. 修改一个点的权值
 3. 对于一个点 x ，询问所有与该点路径上颜色相同的点中,权值的最大值
- $n, m \leq 10^5$

例题21

- 相比上题多用一个 $multiset$ 维护虚边最大值即可。

总结

- *LCT*能解决链修改，链询问，以及一部分子树询问，但是部分子树修改并不能支持，如果想了解子树修改怎么搞，可以去研究一下*ETT*和*TopTree*两种复(鬼)杂(畜)的数据结构。



感谢聆听