数据结构讲义

LCT部分

1. P4299

   在 X 星球上有 n 个国家，每个国家占据着 X 星球的一座城市，城市从 1 至 n 编号。由于国家之间是敌对关系，所以不同国家的两个城市是不会有公路相连的。

   X 星球上战乱频发，如果 A 国打败了 B 国，那么 B 国将永远从这个星球消失，而 B 国的国土也将归 A 国管辖。A 国国王为了加强统治，会在 A 国和 B 国之间修建一条公路，即选择原 A 国的某个城市和 B 国某个城市，修建一条连接这两座城市的公路。

   同样为了便于统治自己的国家，国家的首都会选在某个使得其他城市到它距离之和最小的城市，这里的距离是指需要经过公路的条数，如果有多个这样的城市，编号最小的将成为首都。

   现在告诉你发生在 X 星球的战事，需要你处理一些关于国家首都的信息，具体地，有如下3种信息需要处理：

   - `A x y`：表示某两个国家发生战乱，战胜国选择了 x 城市和 y 城市，在它们之间修建公路（保证其中城市一个在战胜国另一个在战败国）。
   - `Q x`：询问当前编号为 x 的城市所在国家的首都。
   - `Xor`：询问当前所有国家首都编号的异或和。

   $1 \leq n \leq 10^5$，$1 \leq m \leq 2 \times 10^5$，$1 \leq x, y \leq n$。

   【解答】

   动态维护树的重心
   维护子树信息是不会少的。在此基础上，网上大多数解法都是启发式合并。利用了以重心为根的子树大小不超过原树的一半，每次合并两颗树的时候，小的往大的上面合并，而且是一个点一个点地link上去，每次link完检查一下这个子树如果超过了当前整个树大小的一半，就把重心向当前点移动一下。这样做，合并次数上限$O(N \log N)$，每次更新重心也需要$O(\log N)$的复杂度，所以复杂度极限是$O(N \log^2 N)$，还不够优秀。
   因此我就想再优化一下算法。复杂度瓶颈就在于一个一个link，能不能考虑直接连接两颗树对重心的影响呢？
   答案是肯定的。又要用到一个性质——连接两颗树后，新的重心一定在原来的两个重心的路径上。这个证明不难，反证一下就好了。
   那就可以直接link然后把链提出来去找了。我们当然不能把整条链全找遍了，毕竟重心的性质那么多，也要好好利用嘛。
   具体找法：类似树上二分，我们需要不断逼近树的重心的位置。记下lsum表示当前链中搜索区间左端点以左的子树大小，rsum表示右端点以右的。x的整个子树就表示了当前搜索区间，在中序遍历中x把搜索区间分成了左右两块（在Splay中对应x的左子树和右子树）。
   如果x左子树的s加上lsum和x右子树的s加上rsum都不超过新树总大小的一半，那么x当然就是重心啦！当然，如果总大小是奇数，重心只会有一个，那就找到了。否则，因为必须编号最小，所以还要继续找下去。
   当我们没有确定答案时，还要继续找下去，那么就要跳儿子了。x把整个链分成了左右两个部分，而重心显然会在大小更大的一部分中，这个也应该好证明。如果x左子树的s加上lsum小于x右子树的s加上rsum，那就跳右儿子继续找。这时候当前搜索区间减小了，搜索区间以外的部分增大了，lsum应该加上s[l]+si[x]+1。反之亦然。如果跳进了空儿子，那肯定所有情况都考虑完了，直接结束查找。（其中的s和si对应的是上面的siz和siz2）
   当然，重心找到了就还是要伸展一下，保证复杂度。（蒟蒻造数据的时候懒得去卡了qwq）

这一部分套用Splay的复杂度，是均摊$O(\log N)$的，总复杂度也就降到了$O(N \log N)$。
findroot实在很慢，于是可以写个并查集来维护每个点所在树的重心。

2. P4338

这个世界有 n 个城市，这 n 个城市被恰好 $n-1$ 条双向道路联通，即任意两个城市都可以 互相到达。同时城市 1 坐落在世界的中心，占领了这个城市就称霸了这个世界。

在最开始，这 n 个城市都不在任何国家的控制之下，但是随着社会的发展，一些城市会崛 起形成国家并夺取世界的霸权。为了方便，我们标记第 i 个城市崛起产生的国家为第 i 个国家。 在第 i 个城市崛起的过程中，第 i 个国家会取得城市 i 到城市 1 路径上所有城市的控制权。

新的城市的崛起往往意味着战争与死亡，若第 i 个国家在崛起中，需要取得一个原本被国 家 $j(j \neq i)$ 控制的城市的控制权，那么国家 i 就必须向国家 j 宣战并进行战争。

现在，可怜知道了，在历史上，第 i 个城市一共崛起了 a_i 次。但是这些事件发生的相对顺 序已经无从考究了，唯一的信息是，在一个城市崛起称霸世界之前，新的城市是不会崛起的。

战争对人民来说是灾难性的。可怜定义一次崛起的灾难度为崛起的过程中会和多少不同的国家进行战争（和同一个国家进行多次战争只会被计入一次）。可怜想要知道，在所有可能的崛 起顺序中，灾难度之和最大是多少。

同时，在考古学家的努力下，越来越多的历史资料被发掘了出来，根据这些新的资料，可怜会对 a_i 进行一些修正。具体来说，可怜会对 a_i 进行一些操作，每次会将 a_x 加上 w。她希望 在每次修改之后，都能计算得到最大的灾难度。

然而可怜对复杂的计算并不感兴趣，因此她想让你来帮她计算一下这些数值。
对题面的一些补充：

- 同一个城市多次崛起形成的国家是同一个国家，这意味着同一个城市连续崛起两次是不会 和任何国家开战的：因为这些城市原来就在它的控制之下。
- 在历史的演变过程中，第 i 个国家可能会有一段时间没有任何城市的控制权。但是这并不 意味着第 i 个国家灭亡了，在城市 i 崛起的时候，第 i 个国家仍然会取得 1 到 i 路径上的城市的控制权。

$n, m \leq 4 \times 10^5, a_i, w_i \leq 10^7$

【解答】

首先要想到30分的结论
已知树上每个点access的总次数，构造出一个顺序，最大化虚实边的切换总次数
其实如果能发现最优顺序的构造是没有后效性的话，问题便可以进一步简化
考虑每个点的子树。假设已经对所有子树中的点构造出了一个最优顺序（一个序列），那么一定不会和它的所有祖先的子树中的最优序列产生冲突。这个并不好证明，仔细想一想应该能发现。
于是就可以单独考虑每个点x。能对x的实子边产生影响的是x的所有子树和x本身（access(x)会使x没有实子边），每切换一次都会使答案$+1$。显然同一个子树中产生的影响是相同的。于是我们要让来自不同子树（或x本身）尽可能交替access。当没有某个子树（或x本身）的a总和过大时，可以构造出使得（除了第一次）每一次access都有贡献的方案。如果某个子树（或x本身）的a总和过大，大于所有子树总和的一半时，是不可以的，那个子树（或x本身）的某几次操作肯定不会有贡献。用数学公式大概表示每个点的贡献为（S为子树的a之和，c为x的每个子树）
$\min\{S_x - 1, 2 * (S_x - \max\{a_x, \forall S_c\})\}$
当$\max\{a_x, \forall S_c\} > \frac{S_x+1}{2}$时$\min$取后者
用树形DP算出来就有30分了
那么怎样快速修改呢?
首先，对某个点的a加上一个值w，只可能会影响该点到根的路径上的点的贡献。
因为是加一个值，所以假如这些点中某些点的子树S大于它父亲子树S的一半，那么$S_x + w$，$\max\{a_x, \forall S_c\}$也会$+w$，带入上式发现贡献是不变的!
看到某个子树S大于所有子树总和的一半，有没有想到树剖？树剖的轻重边就是这样划分的啊!

同样对维护好每个点子树S的树进行轻重链剖分，某些点的子树a之和大于它父亲子树a之和的一半就连重边（否则连轻边），这样每个点至多有一个重儿子。类似树剖的证明，每个点到根的轻边总数（也就是我们可能会修改的点数）是$\log \sum a$级别的！
修改的复杂度也有保障啦！我们只要快速找到这些点就好了。用实链剖分维护子树信息即可。全局保存ans，每次进行类access操作找到虚边，就让ans直接减去以前的贡献，加上w以后判断当前的情况决定是否要切换虚实边并让ans加上新的贡献即可。
为了方便计算以前的贡献，蒟蒻觉得可以保存一下以前贡献的类型（无非就三种，某子树过大、自己过大、都不是很大）算的时候就省去了一些判断的时间。

后面也可以用树剖+线段树二分之类的方法找到虚边并维护。

3. P3703

Bob 有一棵 n 个点的有根树，其中 1 号点是根节点。Bob 在每个点上涂了颜色，并且每个点上的颜色不同。

定义一条路径的权值是：这条路径上的点（包括起点和终点）共有多少种不同的颜色。

Bob可能会进行这几种操作：

- `1 x` 表示把点 x 到根节点的路径上所有的点染上一种没有用过的新颜色。
- `2 x y` 求 x 到 y 的路径的权值。
- `3 x` 在以 x 为根的子树中选择一个点，使得这个点到根节点的路径权值最大，求最大权值。

Bob一共会进行 m 次操作

$1 \leq n \leq 10^5$，$1 \leq m \leq 10^5$。

【解答】

首先 **操作**1 具有一个特性，染色操作全部都是直接染到顶点。
让人想到 LCT 的 $access$ 操作。
发现每次都是开一个新的颜色，我们可以知道：一个点到根节点有多少种颜色，体现在 LCT 上即：从此点走到根节点需要走的虚链个数 $+1$。
初始时每个点都是一个新颜色，即 LCT 中的所有边都是虚边。
考虑记$dis[x]$表示从根节点走到此点需要经过多少条虚链$+1$，那么最初的 $dis[x]$ 为 $dep[x]$
相对而言较为好处理的是 2 操作，考虑树上差分，倍增求 LCA 不难发现答案为：
$dis[x]+dis[y]-2*dis[lca]+1$
接下来考虑操作3 操作3 需要维护的是子树信息。
所以需要用一个数据结构来维护 dis数组 $\rightarrow$ 线段树。
如果对原树按照 dfs 进行遍历，在 dfs 序中每棵子树对应的区间都是连续的。
那么操作3就变成了在线段树中询问区间最大 dis 值。
现在我们来考虑最恶心的 **操作**1
这是一个 $access$
我们看看一般的$access$是怎么写的：

```
void access( int x ) {
    for( int y = 0; x; y = x, x = t[y].fa )
        Splay(x), t[x].son[1] = y, pushup(x);
}
```

如果将这段代码分得更细一点，其实是：
1.先将 x 旋到$Splay$根。
2.将 x 的右儿子变虚
3.将 x 的虚儿子(y) 变实。
然后$pushup$
我们发现这一过程中其实发生了：1.一条虚链变成了实链，2.一条实链变成了虚链。

考虑它原来的儿子，这条实链变成了虚链，那么它的儿子及其下面所有的点往根节点走的时候都要多经过一条虚链，换而言之：此儿子所管辖的子树区间的 dis 值全部 $+1$
然后另一个操作为：一个虚儿子变成了实儿子，那么此儿子变成了实儿子，那么其的儿子走到根节点的路上都会少走一条虚链，等价于：此儿子所管辖的所有子树区间的 dis 值全部 -1
注意坑点：比如这样写就是错的（伪代码）：

```
void access( int x ) {
    for( int y = 0; x; y = x, x = t[y].fa ) {
    Splay(x), update( rs(x), + 1 ); //更新右儿子。
    update( y, -1 ), t[x].son[1] = y;
}
```

需要注意：1.LCT 中的$Splay$作为辅助树，其维护的是一个中序遍历，并不是真实树中的一种父子关系，然而我们做的操作是对其真实的儿子所管辖的区间做的$+1/-1$ 所以我们要找到其原本的儿子。
所以我们对于原树，我们需要找到其右子树(深度比其大的点)中深度最小的点。
对于接上去的点，我们也需要找到其中深度最小的点。
故其实还要一个 $findroot$ 函数（某一颗子树中深度最小的点）

还有一个纯树剖解法，简单写一下：

称连续一段相同颜色点组成的链叫同色链
开线段树，mx记录该区域内点到根路径中最大权值 （最大点权）
col记录该区域最新被赋过的颜色（颜色编号开始为1-n，后从n+1开始递增）
则一个点颜色实际颜色为含有该点区间中col值最大的（编号越大颜色越新）
对于操作2，易证明符合可减性，故可以用求两点间距离的方法求
$(ans(u,v)=dis[u]+dis[v]-2*dis[lca(u,v)]+1)$
对于操作3，线段树查询mx的最值
最难的是操作1：
从x开始向上跳，每次跳过一条重链或同色链（哪个起点更深就跳那个）
跳的时候区间修改颜色，更新点权
（利用跳的这一段颜色相同，注意不要把上次跳的链也更新了）
还要注意更新原来的同色链的最顶端
复杂度 $O(n\log^2 n)$

4. P5385

你有一个无向图 $G(V,E)$, E 中每一个元素用一个二元组 (u,v) 表示。

现在把 E 中的元素排成一个长度为 $|E|$ 序列 A。

然后给你 q 个询问二元组 (l,r),

表示询问 图 $G'\left(V,\bigcup\limits_{i\in[l,r]}\{A_i\}\right)$ 的联通块的个数。

$|V|\le 10^5,|E|\le 2*10^5,q\le 10^5$，强制在线

【解答】

首先我们知道 n 个点的树有 $n-1$ 条边，因此对于森林来说，其点数减边数即为树的个数。那么对于普通的图，求出其任意一个生成树森林，森林中树的个数即为原图中连通块的个数，也就是点数减边数。
因此问题就转化为了如何快速求出一个图的生成树森林的边数。
考虑用 LCT 来维护原图的一个生成树森林。按顺序加边，当发现两端点已经连通，要形成环时，就删去环上最早加入的一条边。同时用主席树来维护每条边是否在当前的生成树森林中出现。
询问时在 r 所对应的主席树上查询区间 $[l,r]$ 中有几条边存在，存在的边数即为对应的生成树森林

的边数，用 n 减去边数即为答案。
因为 LCT 维护的是以时间为边权的最大生成树森林，所以每条边都尽可能的选用出现时间晚的，这就使得 r 所对应的主席树中区间 $[l,r]$ 中边的出现是最多的，所以就保证了正确性。

5. P9201

给你 n，表示一共有 n 位赛博佛祖，编号依次为 $1\sim n$。

第 $i(1\le i\le n)$ 位赛博佛祖可以对应为一个二元组 $\langle S_i,d_i\rangle$，其中 S 在任意时刻均为 $\{1,2,3,\dots,m\}$ 的一个子集（可以为空），而 d_i 为 $1\sim m$ 间的整数。

如果在某一时刻，存在一位赛博佛祖的 S_i 为空集，佛祖会感到很开心而给你加功德。具体地，他会敲响第 d_i 个木鱼，并**在下一时刻同时**影响所有的 n 位赛博佛祖（包括他自己）。对第 $j(1\le j\le n)$ 位赛博佛祖，如果 $d_i\in S_j$，那么将从 S_j 内删去 d_i；否则向 S_j 内加入 d_i。如果有多位赛博佛祖的 S_i 为空集，取编号最小的 i 为你加功德。

现在作为电子资本家的你，想要功德无量。你想知道，最少再请来几位赛博佛祖，可以使得你的这些佛祖们**源源不断地**为你加功德。假设这个答案是 s（可以为 0），那么新的佛祖们的编号依次为 $(n+1)\sim(n+s)$。

因为你是个资本家，有时候你不想请那么多佛祖。所以你有许多组询问，对于一组 l,r，设 $f(l,r)$ 表示如果初始只有 $[l,r]$ 之间的佛祖，答案将会是多少，注意，每组询问相互独立，上一次添加的佛祖不会延续到以后的询问中。

为了避免太多的输出，你只需要输出：

$$\sum_{l=1}^{n}\sum_{r=l}^{n}f(l,r)\times 2^l$$

即可，答案对 10^9+7 取模。

$1\le n\le 10^5$，$1\le m\le 17$。

【解答】

先考虑单个询问。
状压集合，然后操作变成每个 S 异或上 2^{d_*}。
由于是全局异或，显然只需要关心目前异或的总量。
并且，对一个总异或和为 p 的局面，取出的元素是确定的（即编号最小的 $S=p$ 的元素），新局面是**唯一**的，如果取不出来那么操作结束，这是我们所需要避免的。因此可以对局面之间的转移**建图**（点集为 $\{0,1,\cdots 2^m-1\}$），用有向边刻画关系，形成一个**内向树或内向基环树森林。**
思考加一个元素的直观感受，显然新加元素的 S 不会与之前的某个 S 相同（因为永远不会用到它），因此新加一个元素可以看做**选择一个无出边的点，加一条出边**，且指向的点和它差恰好一个二进制位。
而操作能无限进行，就相当于**从 0 出发能到一个环。**
现在可以考虑求解答案了：

- 0 处于一个内向基环树中，那么答案为 0。
- 0 不在内向基环树中：
  - 0 所在弱连通块大小 ≥ 2，那么答案为 1。
  - 0 为一个孤点：
    - 若某个 $\text{popcnt}=1$ 的点在一个内向基环树中，那么答案为 1。
    - 否则，答案为 2。

这样，枚举 l，然后从小到大枚举 r，维护一个并查集即可。
然后考虑正解。
l 从大到小扫描线，S 相同的体现为修改出边指向的节点。
那么我只需要分别求出从 l 开始加边，0 以及每个 $\text{popcnt}=1$ 的节点在某个内向基环树的**最小右**

端点。

设第一部分为 T_1，第二部分的 $\min$ 为 T_2，那么：

- $r \in [l, T_2 - 1]$，同时 $[l, r]$ 内不存在与 0 相关的连边，这部分贡献 1。
- $r \in [l, T_1 - 1]$ 这部分也贡献 1。

（注意我把上面答案为 2 的情况拆成两部分贡献了）

于是考虑怎么维护右端点，每个点记录一个点权表示它在序列中的下标，那么相当于从某个起点出发走到环，经过点权的 $\max$（如果不会走到环那么记为 $n+1$）。

于是变成了一个类似加边，删边，查到根路径点权 $\max$ 的问题，唯一区别在于变成了基环树，我们强行改造 LCT 使它能维护这个问题。

对于每个弱连通块维护一个**有根树**结构：

- 若它为内向树，以无出边的那个点为根。
- 若它为内向基环树，以环上任意一个点为根，并记录其出边指向的节点，称这条边为**附加边**。

再具体讨论一下加删边的细节：

删边：删除 u 的出边

- u 为所在弱连通块的根，那么直接把附加边去掉即可。
- u 不为根，如果弱连通块为内向树则直接断掉，如果是内向基环树，要分是否为环边进行讨论。
  - 首先确定是否为环边，记附加边的另一端为 x，那么相当于判定 u 是否在 x 到根路径上，这个可以通过 `access(x),splay(x)` 打通 x 到根的路径，然后从 u 开始跳到所在 splay 的根，判是否与 x 相同即可。
  - 如果是环边，那么断掉它，连上附加边，同时把**树根定为** u。
  - 如果不是环边，那么正常断边。

加边：加上 u 的出边 $u \to t$

- u, t 在同一个弱连通块内，则**把树根定为** u，并记录附加边。
- 否则，正常连边，连边后整个弱连通块的根定为原 t 所在弱连通块的根。

注意 `Link`，`Cut` 均有可能改变树根，因此在做完 LCT 操作后都要注意恢复正确的树根。.

最后一部分是查询：

- 先查出 u 所在弱连通块的根，若其没有附加边则为内向树，返回 $n+1$。
- 否则，求出 u 到根路径的点权 $\max$，和环上点权 $\max$，取大的那个返回即可。

时间复杂度为 $O(nm \log n)$。

6. CF1109F

给一个$n \times m$的网格图（$n, m <= 2000, nm <= 2 * 10^5$），每个格子上有个权值$f_{ij}$，保证$f_{ij}$构成一个1~$nm$的排列。问有多少区间满足这个权值在这个区间内的格子构成的连通块是一棵树。

【解答】

考虑一颗树的性质：

- 点数减边数等于 1。
- 没有环。

当一张图满足第二点，那么其点数减边数大于等于 1。

回到本题，考虑对 l 扫描线。对于一个 l，找到一个最大的 r，使得 $[l, r]$ 构成的连通块不形成环。

这个显然可以 LCT + 双指针解决，因为对于 $l < l' \le r$，若 $[l, r]$ 满足第二点，那么 $[l', r]$ 一定满足。

然后就是统计多少个 $x \in [l, r]$，使得 $[l, x]$ 构成的图，点数减边数等于 1。

这个可以用线段树解决：对于每个叶子节点 $[i, i]$，记录 $[l, i]$ 构成的图的点数 - 边数，还要支持询

问有多少个 1。注意到对于任意叶子节点其值都不小于 1，故问题转化为求最小值及其个数。
当 $l \leftarrow l+1$ 时，只需要在线段树上删除这个点和这个点所连边的影响即可。

7. CF1172E

给一颗大小为 n（$2 \leq n \leq 4 \cdot 10^5$） 的树，每个点有一个颜色 c_i（$1 \leq c_i \leq n$）

定义 $dis(u,v)$ 为点 u 到 v 的路径上的颜色数。

现在希望你求出：

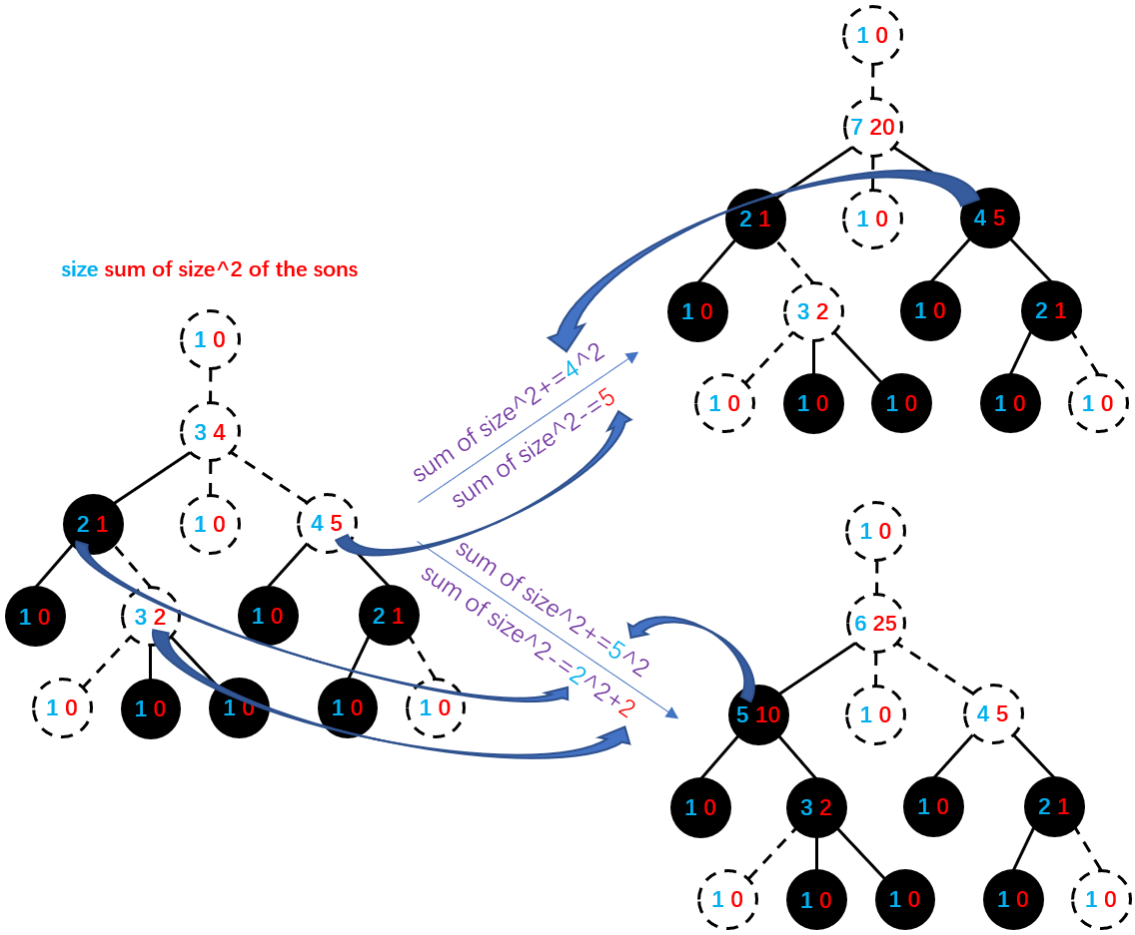
$\sum_{i=1}^{n} \sum_{j=1}^{n} dis(i,j)$

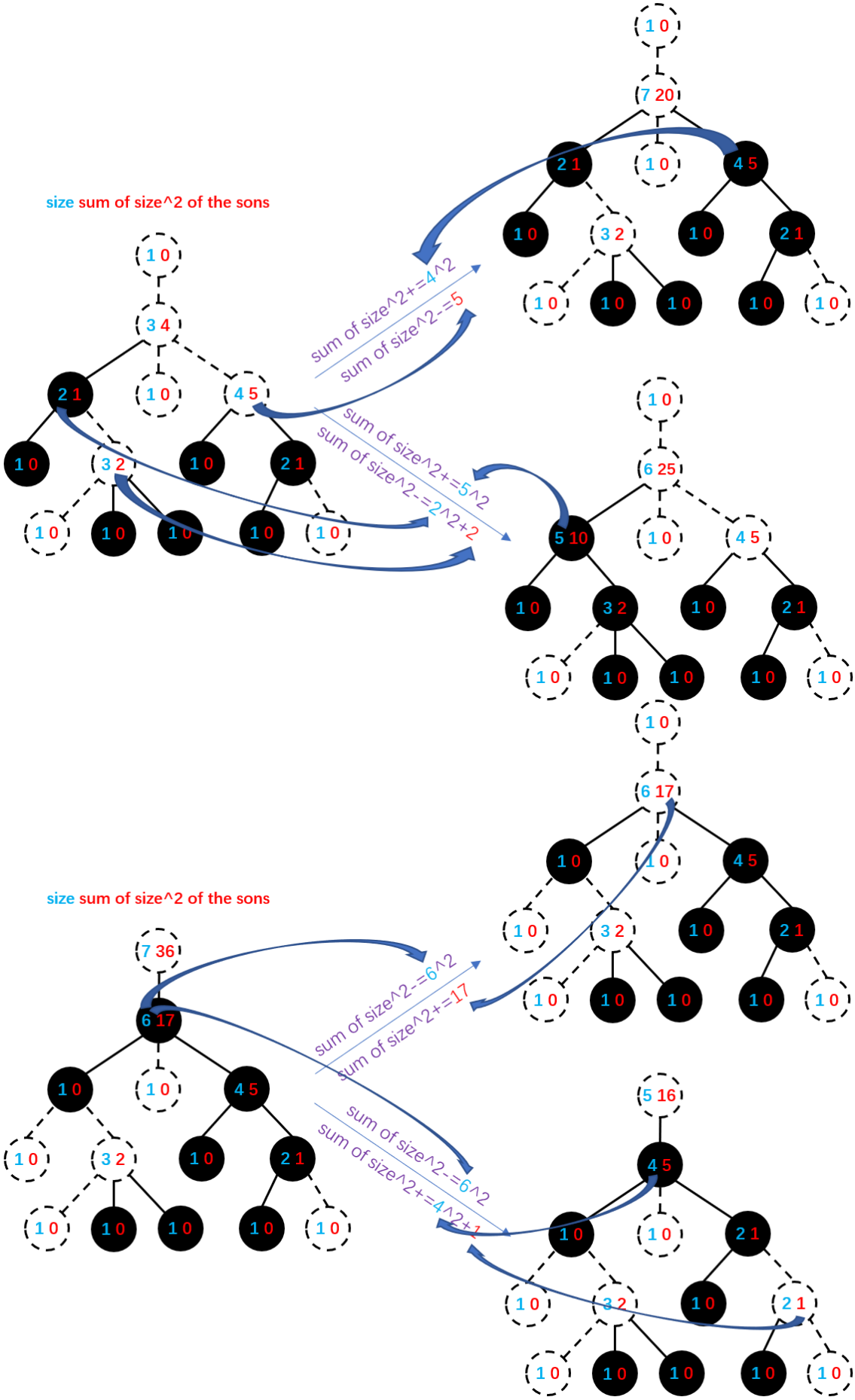
特别的，我们有 m （$1 \leq m \leq 4 \cdot 10^5$） 次修改，每次修改一个点的点权，每次修改后，你都需要重新输出上值。注意，你仍然需要输出初始的答案。

所以你的输出应当为 $m+1$ 行。

【解答】

对每种颜色分别考虑不含该颜色的简单路径条数。
令当前处理的颜色为 c，把颜色为 c 的视为白色，不是 c 的视为黑色，那么不含 c 的路径条数就是每个黑联通块的大小的平方和，修改就是当颜色是 $c \leftrightarrow$ 颜色不是 c 时翻转一个点的颜色。所以，问题转化成了黑白两色的树，单点翻转颜色，维护黑联通块大小的平方和。这个转化后的问题可以用很多数据结构做，这篇题解里使用 Link/cut Tree.
对每个点维护子树大小，儿子大小平方和，在 link/cut 的时候更新答案即可。有一个~~大家熟知的~~ trick，就是每个黑点向父亲连边，这样真正的联通块就是 Link/cut Tree 里的联通块删掉根。
具体看图吧，图讲的挺清楚的：

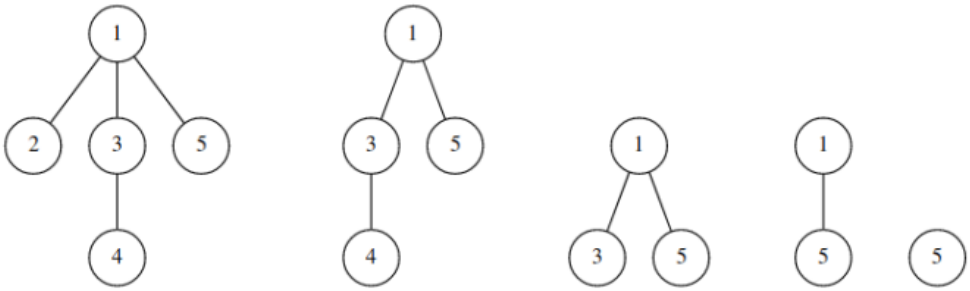




LCT 的部分就是这样，计算答案的时候先初始化一个全是黑点的树，离线处理每个颜色，处理完一个颜色反着改回去。

8. CF1137F

我们定义一棵树的删除序列为：每一次将树中编号最小的叶子删掉，将该节点编号加入到当前序列的最末端，最后只剩下一个节点时将该节点的编号加入到结尾。



例如对于上图中的树，它的删除序列为：2 4 3 1 5

现在给出一棵n个节点的树，有m次操作：

`up v`：将v号节点的编号变为当前所有节点编号的$\max+1$

`when v`：查询v在当前树的删除序列中是第几号元素

`compare u v`：查询u和v在当前树的删除序列中谁更靠前

$n,m\leq 200000$

【解答】

考虑将一个点 x 的编号变为当前所有点编号最大值 $+1$ 会对每个点的删除时间产生怎么样的影响。由于编号最大的点肯定是最后一个被删除的，因此我们不妨令编号最大的点为根，那么可以发现，对于不在 x 到根这条路径上的点，它们删除的相对位置顺序是不会发生变化的，因为删除这样的点时，肯定它们的儿子已经被删除了，而它的父亲肯定没被删除，因此 x 到根节点这条路径上的点的删除顺序肯定不影响其它点的删除顺序，而由于此时 x 和原来的根节点 r 分别是所有点中，权值最大和第二大的点，因此 $x\to r$ 中的点肯定是最后一批被删除的，并且删除顺序取决于它们与 r 的远近。

思考如何维护这个东西，发现这东西有点类似于 P3703 树点染色 的模型，修改一个点点权时，相当于将这个点先 `access` 一遍把 $x\to r$ 这条链搞出来并赋成一个新的颜色，然后再 `makeroot(x)`。因此我们尝试用 LCT 维护这个东西，对于每个实链，我们给它一个优先值 v，优先值越大表示这条链上的点越靠后被删除，而同一条实链上的点的删除顺序则是按照深度的增加而递减，因此考虑建一个 BIT，对于每种优先级 v 维护当前有多少个点的优先级为 v，这样查询就做一遍前缀和，减掉当前实链中深度比当前节点小的点数，这个可以通过 LCT 的 siz 维护（注意，这里不需要写 top tree，因为是维护链的信息）。BIT 信息的维护就按照套路在 LCT 时扣掉除了右子树之外部分原来优先级的贡献，加上新贡献即可。

时间复杂度 $n\log^2 n$

数据结构题目选讲

1. CF1436E

   求一个数列的所有子区间的 **mex** 值的 **mex**

   某个数组的 **mex** 是这个数组中没有包含的最小正整数。

   $n\leq 10^5,a_i\leq n$

   【解答】

   我们只要依次去看mex为$1,2,\ldots$的子区间是否存在。我们看mex为k的区间是否存在的时候，我们考虑序列中所有相邻的k，只要去看相邻的两个k直接的区间$[l,r]$中是否出现过所有的$[1,k-1]$的数。

那么我们可以从前往后扫整个序列，对每个值去记录最后出现的位置，当扫到一个值为k的数的时候，我们去查询$1\sim k-1$最后出现位置的最小值，看看是否大于k上次出现的位置。

复杂度$O(n\log n)$

2. CF1425I

Chanek先生有一个果园，果园的结构是一棵三叉的有根树。树上有N个顶点，编号从 1 到 N ，并且树的根是顶点 1 。对于 $2 \leq i \leq N$ ，定义 P_i 表示顶点i的父节点。树的高度不大于 10，这里的树的高度被定义为树根到树上顶点的最大距离。

果园的每个顶点上都存在一个灌木丛。最初所有的灌木丛上都有果实，顶点 i 的灌木丛在最后一次被采摘后的 A_i 天后将会长出果实。目前已经有果实的灌木丛上不会再长出果实。

Chanek先生会到他的果园参观 Q 天。在第 i 天，他将采集节点 X_i 的子树内所有灌木上的果实。你需要求出每一天所有被采集的灌木到 X_i 的距离之和，以及当天被采集的果实的数量。

$N, Q, A_i \leq 5 \times 10^4$

【解答】

显然我们需要统计子树内果子的个数以及深度之和即可计算答案。但是我们不可能把所有的果子都枚举一遍，因为**有的**点的a_i比较小，枚举会炸掉。这样的话我们可以想到对a_i按照根号分类讨论：
①对于$a_i>\sqrt{q}$的点，我们可以$O(\sqrt{q})$枚举它长出的所有果子，就是找长出果子后在最早的哪天被摘掉。这个可以离线dfs，维护每个点到祖先的一条链上的所有询问，对于每个果子，在存着祖先询问的线段树或set上二分摘掉日期即可。这是$O(n\sqrt{q}\log n)$的。
②对于$a_i\leq\sqrt{q}$的点，$O(\sqrt{q})$枚举a_i，我们把a_i相同的都扔进一棵线段树中（我们发现原树就是一棵线段树的样子，这里我们必须拿原树的线段树结构来做，不然无法保证复杂度）。对于每个节点，我们去存储一个值r_i表示这个节点的灌木还需要r_i秒才能结果。然后对于每个节点再去存储max_i和min_i分别表示子树中r_i的最大值和最小值。一开始每个点的r_i都为0。接下来依次考虑每个询问X，做以下操作：1. 对于X内所有$max_i=0$的节点，我们结算贡献并且打上标记a_i表示所有内部所有节点都被修改为a_i了。（并且不递归进去）；2.把所有的r_i改为$\max(r_i-1,0)$。待会我们会证明操作的复杂度为均摊的$O(\log n)$，这样这部分的复杂度也为$O(n\sqrt{q}\log n)$。

复杂度证明：我们令$mark$为子树r_i的最大值，并且规定如果一个节点有$mark$为M，它的所有子树中的值为M的$mark$都被删除。令$F(X)$为X子树中0这个$mark$的数量，令$P=\sum F(X)$，那么一开始$P=F(1)=1$。接下来我们来看对于子树X的操作对P的变化：

1. 我们给子树X打标记a_i的过程，我们不会增加P的值。并且我们会发现，任意时刻任意的非0的$mark$只会在树上出现一次。
2. 在一个$mark$从1变成0的时候，P最多增加$\log n$。因为1这个$mark$只有一个，所以显然。
3. 每个额外节点的访问至少会让P减少1

所以我们可以得出复杂度是正确的。

3. CF1491H

给定一棵大小为 n 的 1 为根节点的树，树用如下方式给出：输入 $a_2, a_3, \ldots, a_n$，保证 $1 \leq a_i < i$，将 a_i 与 i 连边形成一棵树。

接下来有两种操作：

- `1 l r x` 令 $a_i = \max(a_i - x, 1) (l \leq i \leq r)$。
- `2 u v` 查询在当前的 a 数组构成的树上 u, v 的 LCA。

两种操作共有 q 个。

$2 \leq n, q \leq 10^5$，$2 \leq l \leq r \leq n$，$1 \leq x \leq 10^5$，$1 \leq u, v \leq n$

【解答】

钦定树根为 1，那么所有的连边都是小编号节点往大编号节点连边。
考虑分块。将 a 序列分成 $\sqrt{n}$ 块，每块有 $\sqrt{n}$ 个数，记每一块的左边界为 L_i，右边界为 R_i。pos_i 表示节点 i 所属的块。
接下来是比较神仙的一步。定义 b_i 为**节点 i 的祖先中编号最大的与 i 不在一个块的节点编号**。考虑如何在 a_i 更新时重构一个块 x 的 b_i 值。对于这个块 x，我们可以从左往右枚举节点 i，判断其 a_i 是否小于 L_x，如果是的话则 $b_i=a_i$，否则节点 i 的 b 值就可以转移到节点 a_i 的 b 值，即 $b_i=b_{a_i}$。
首先看修改操作，直接考虑整块和零散点怎么处理。
对于零散点，直接暴力修改其 a_i 值，然后重构整个块的 b 值即可。
而对于整块，我们就需要打懒标记 f_i 表示其 a_i 还没减多少。可这个时候我们发现整块的 b 已经变了，对后面的查询操作就有影响，而每次暴力修改 b 需要遍历整个块，时间复杂度便无法保证。但我们可以注意到，对于每个块 i，它经过 $\sqrt{n}$ 次修改操作后，a_i 都至少会变为 $a_i-\sqrt{n}$，此时节点 i 的所有祖先**都一定在这一块之前**，于是此时的 b_i 值都不变，且为 a_i。
因此，我们可以维护每一块被修改的次数 cnt_i。当 $cnt_i>\sqrt{n}$ 时就直接略过，不需要修改了。
分析一下复杂度，每个块最多修改 $\sqrt{n}$ 次，每次需要 $\sqrt{n}$ 次计算，一共有 $\sqrt{n}$ 个块，于是修改的总时间复杂度就为 $O(n\sqrt{n})$。
其次考虑询问操作，我们发现这个 b_i 很像我们写树链剖分时的 top_i，于是我们可以类比树链剖分进行操作。
对于节点 u,v，我们这样寻找它们的 LCA：

- 当 $pos_u\neq pos_v$ 时，让 pos 值大的那个节点跳到它的 b 值。
- 当 $pos_u=pos_v$ 且 $b_u\neq b_v$ 时，两个节点均跳到它们的 b 值。
- 当 $pos_u=pos_v$ 且 $b_u=b_v$ 时，直接一直让编号大的那个点跳到自己的父亲，直至两点相同即可。

分析一波复杂度。由于一条链上最多有 $\dfrac{n}{\sqrt{n}}=\sqrt{n}$ 次跳 b 操作，而且跳完 b 后每个节点最多只会跳 $\sqrt{n}$ 次，所以单次查询的复杂度是 $O(\sqrt{n})$ 的。
因此，总时间复杂度为 $O(q\sqrt{n}+n\sqrt{n})$

4. CF1017G

给定一棵初始全白的树，维护以下3个操作：

1，1 x表示如果节点x为白色，则将其染黑。否则对这个节点的所有儿子递归进行相同操作

2，2 x表示将以节点x为root的子树染白。

3，3 x表示查询节点x的颜色

$n,q\leq 10^5$

【解答】

首先观察发现，操作的复杂度向修改倾斜，即询问复杂度比较低，可以考虑询问的时候搞点事情。
考虑 (1) 操作的本质。因为要向询问倾斜，所以考虑如果某个点 z 会被 `1 x` 影响到，那么必然有 $x\sim z$ 这条路径上除了 x 之外的所有点都已经被染黑了。
考虑忽略 (2) 操作时，如何通过维护权值实现这个事情。一开始将全部的点权设为 0，每次 (1) 操作就是对该点进行单点 $+1$。这样询问就可以考虑询问 $root\to x$ 这条路径上，是否存在一个后缀 (s,x) 的权和等于 (s,x) 的长度。这样直接做本质上是一个树链上的线段树二分，但其实可以用一个比较传统的技巧优化这个过程，即差分。考虑把每个点的初始点权设为 -1，这样就是询问是否存在一个后缀，其和 $\geqslant 0$。这个可以通过维护链上的最大后缀和来做，写起来简单一些。
然后考虑加上 (2) 操作后如何快速维护。发现 (2) 操作的本质是让整棵子树的点权均变为 -1。但这还不够，因为 (2) 本质上是强制让某个点 x 变成白色，而上面那个做法本质上是对修改离线，最后再分别确定每个修改对答案的贡献。不过也有比较简单的解决方式，即让 `2 x` 中的 x 单点减一个权值 v，使得 $root\to x$ 上的最大后缀和也为 -1。不难发现 $v=qry(x)+1$。
于是最后就可以用 $n\log^2 n$ 的复杂度解决这个问题了。

5. CF1572F

有 n 个广播站，第 i 个广播站高度为 h_i，范围为 w_i。初始 $h_i = 0, w_i = i$。广播站 i 能向广播站 j 传递消息，当且仅当 $i \le j \le w_i$，且 $h_i > \max\limits_{k=i+1}^{j} h_k$。定义 b_i 表示能向 i 传播消息的广播站数量。

接下来有 q 次操作，分为以下两种类型：

- 给出 i, g，把 h_i 变成所有广播站中严格最高的，并且 $w_i \leftarrow g$。
- 给出 l, r，查询 $\sum_{i=l}^{r} b_i$。

$1 \le n, q \le 2 \times 10^5$。

【解答】

直接维护 b_i 非常困难，考虑计算每一个广播站的贡献。设 r_i 表示 i 最远可以传递到哪一个广播站，那么初始 $r_i = i, b_i = 1$。r_i 就表示对 $b_i, \ldots, b_{r_i}$ 有 1 的贡献。每一次修改，c_i 变成最高的，因此 $r_{c_i} = g_i$。对于 c_i 左边的每一个位置 j，一定不能跨过 i，因此 $r_j \leftarrow \min(r_j, i-1)$。可以发现只有单点修改，区间取 min，可以使用吉司机线段树维护 r_i。

接下来考虑 r_i 的变化对 b 的影响。单点修改直接减去原来的值然后加上新的贡献。注意到在吉司机线段树上，区间操作都是把所有最大值 max 变成 v，即修改 cnt_{max} 个值相等的位置。也就是说每一次打 tag 将 max 改成 v 的过程，对应了 b 上 $[v+1, max]$ 这个区间减去 cnt_{max}。因此再用一个线段树维护 b 即可。由于吉司机线段树没有区间加，因此是严格 $O(n \log n)$ 次操作，每次操作需要 $O(\log n)$ 时间在第二棵线段树上区间修改，因此时间复杂度 $O(n \log^2 n)$。