

实连剖分——LCT


主讲：古智锋+程子奇
感谢czq&gzf给予我灵感！
特别鸣谢：广告位招租！

同样将某一个儿子的连边划分为实边，而连向其他子树的边划分为虚边。区别在于虚实是可以动态变化的，因此要使用更高级、更灵活的Splay来维护每一条由若干实边连接而成的实链。

基于性质更加优秀的实链剖分，LCT(Link-Cut Tree)应运而生。

LCT维护的对象其实是一个森林。

在实链剖分的基础下，LCT支持更多的操作

- 
- 1) 查询、修改链上的信息（最值，总和等）
 - 2) 随意指定原树的根（即换根）
 - 3) 动态连边、删边
 - 4) 合并两棵树、分离一棵树
 - 5) 动态维护连通性
 - 6) 更多意想不到的操作 Coming Soon……

LCT和静态的树链剖分很像。怎么说呢？这两种树形结构都是由若干条长度不等的“重链”和“轻边”构成“重链”之间由“轻边”连接。

LCT和树链剖分不同的是，树链剖分的链是不会变化的，所以可以很方便的用线段树维护。但是，既然是动态树，那么树的结构形态将会发生改变，所以我们要用更加灵活的维护区间的结构来对链进行维护，不难想到Splay可以胜任。

- 在这里解释一下为什么要把树砍成一条条的链：我们可以在 $\log n$ 的时间内维护长度为 n 的区间（链），所以这样可以极大的提高树上操作的时间效率。在树链剖分中，我们把一条条链放到线段树上维护。但是LCT中，由于树的形态变化，所以用能够支持合并、分离、翻转等操作的Splay维护LCT的重链（注意，单独一个节点也算是一条重链）。
- 这时我们注意到，LCT中的轻边信息变得无法维护。为什么呢？因为Splay只维护了重链，没有维护重链之间的轻边；而LCT中甚至连根都可以不停的变化，所以也没法用点权表示它父边的边权（父亲在变化）。所以，如果在LCT中要维护边上信息，个人认为最方便的方法应该是把边变成一个新点和两条边。这样可以把边权的信息变成点权维护，同时为了不影晌，把真正的树上节点的点权变成0，就可以用维护点的方式维护边。

LCT的主要性质如下：

性质1： 每一个Splay维护的是一条从上到下按在原树中深度严格递增的路径，且中序遍历Splay得到的每个点的深度序列严格递增。

比如有一棵树，根节点为1（深度1），有两个儿子2, 3（深度2），那么Splay有3种构成方式：

$\{1-2\}, \{3\}$

$\{1-3\}, \{2\}$

$\{1\}, \{2\}, \{3\}$

（每个集合表示一个Splay）

而不能把1, 2, 3同放在一个Splay中（存在深度相同的点）

性质2： 每个节点包含且仅包含于一个Splay中

性质3： 边分为实边和虚边，实边包含在Splay中，而虚边总是由一棵Splay指向另一个节点（指向该Splay中中序遍历最靠前的点在原树中的父亲）。

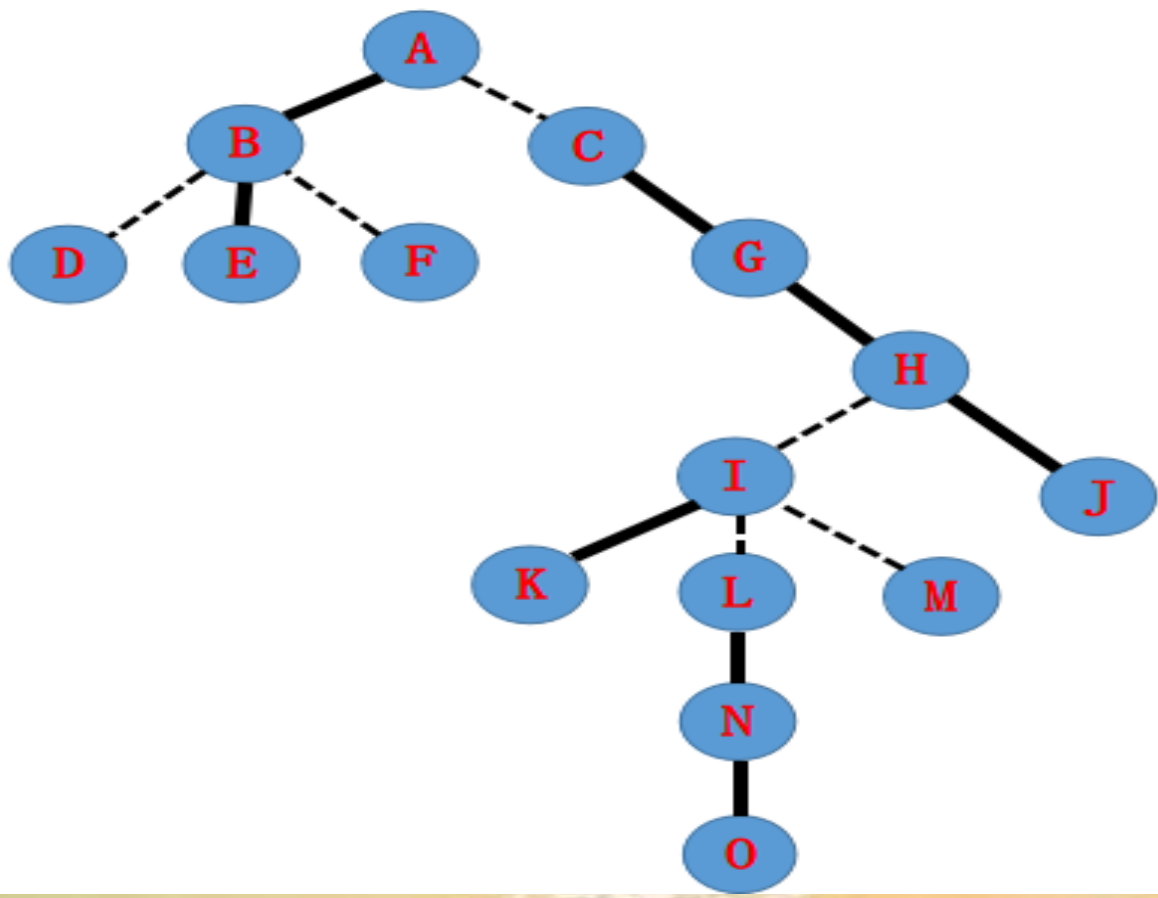
因为性质2，当某点在原树中有多个儿子时，只能向其中一个儿子拉一条实链（只认一个儿子），而其它儿子是不能在这个Splay中的。

那么为了保持树的形状，我们要让到其它儿子的边变为虚边，由对应儿子所属的Splay的根节点的父亲指向该点，而从该点并不能直接访问该儿子（认父不认子）。

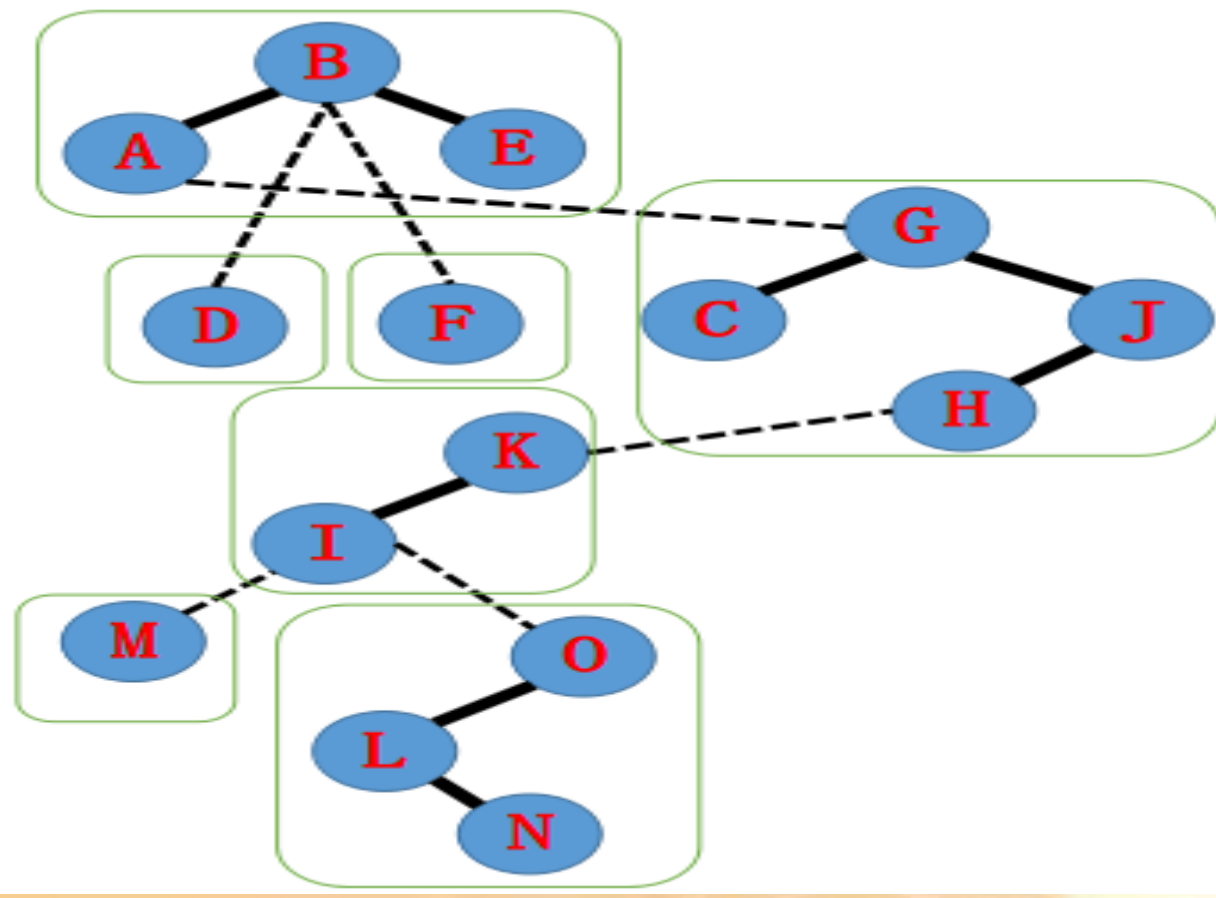
LCT核心操作:access

因为性质3，我们不能总是保证两个点之间的路径是直接连通的（在一个Splay上）。
access即定义为打通根节点到指定节点的实链，使得一条中序遍历以根开始、以指定点结束的Splay出现。

栗子：有一棵树，假设一开始实边和虚边是这样划分的（虚线为虚边）



那么所构成的LCT可能会长这样（绿框中为一个Splay，可能不会长这样，但只要满足中序遍历按深度递增（性质1）就对结果无影响）

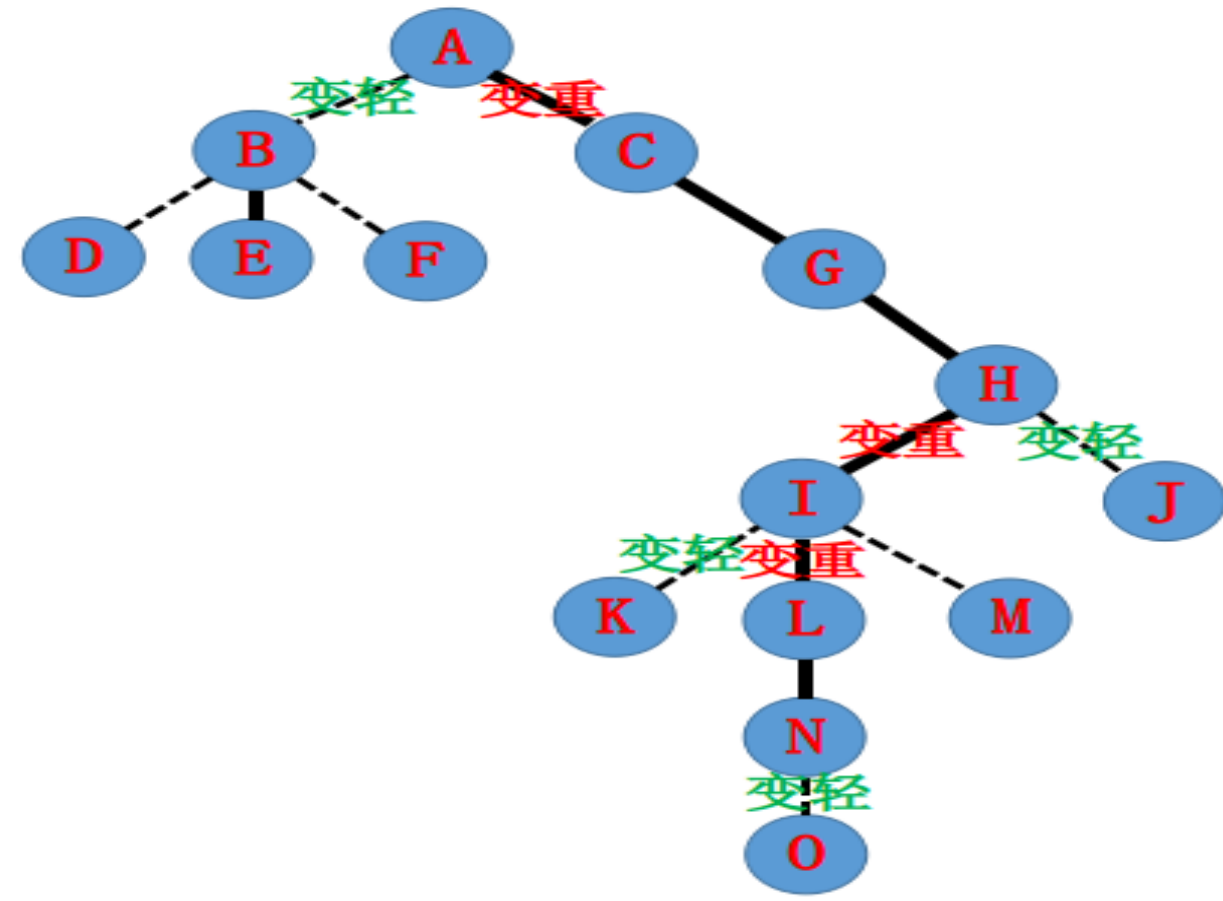


LCT核心操作:access

现在我们要access(N)，把A-N的路径拉起来变成一条Splay。

因为性质2，该路径上其它链都要给这条链让路，也就是把每个点到该路径以外的实边变虚。

所以我们希望虚实边重新划分成这样。



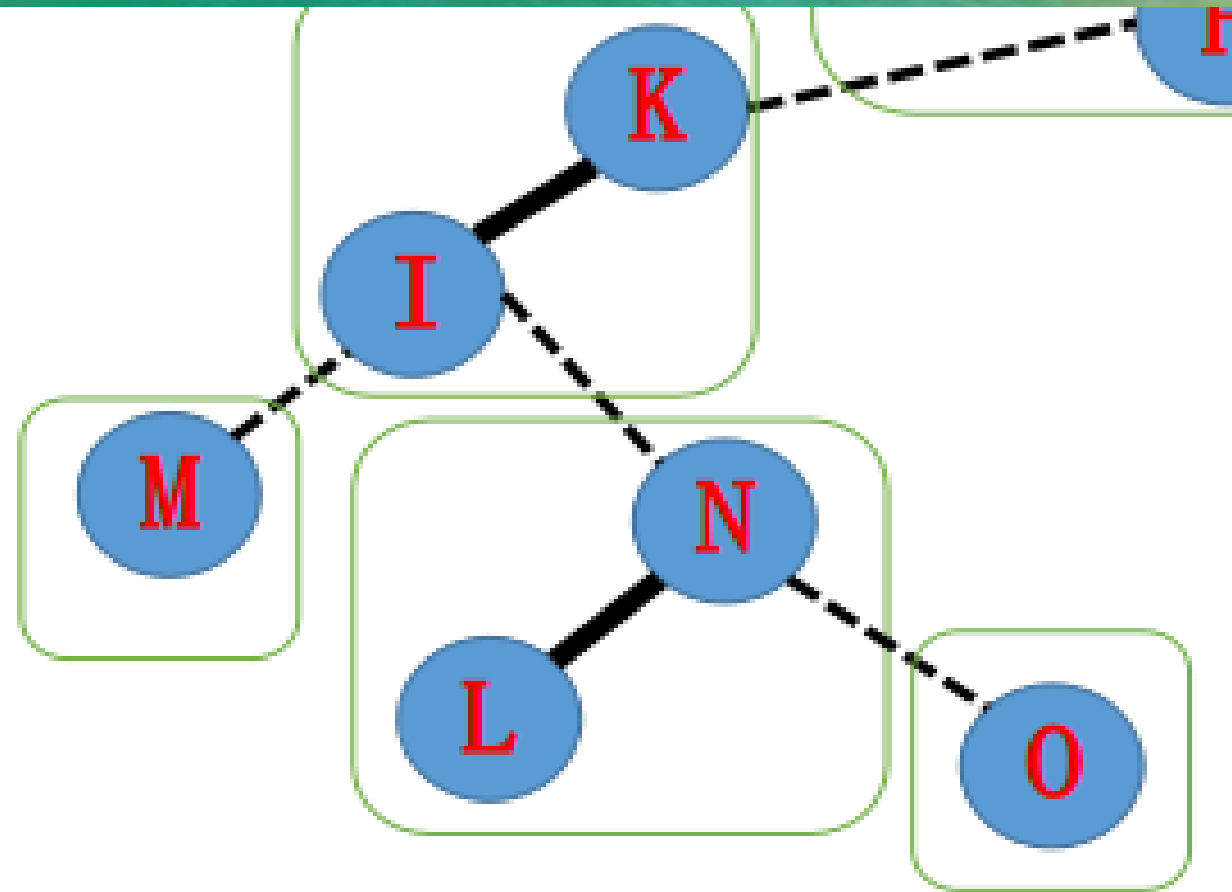
我们要一步步往上拉。

首先把splay(N)，使之成为当前Splay中的根。

为了满足性质2，原来N-0的重边要变轻。

因为按深度0在N的下面，在Splay中0在N的右子树中，所以直接单方面将N的右儿子置为0（认父不认子）

然后就变成了这样——



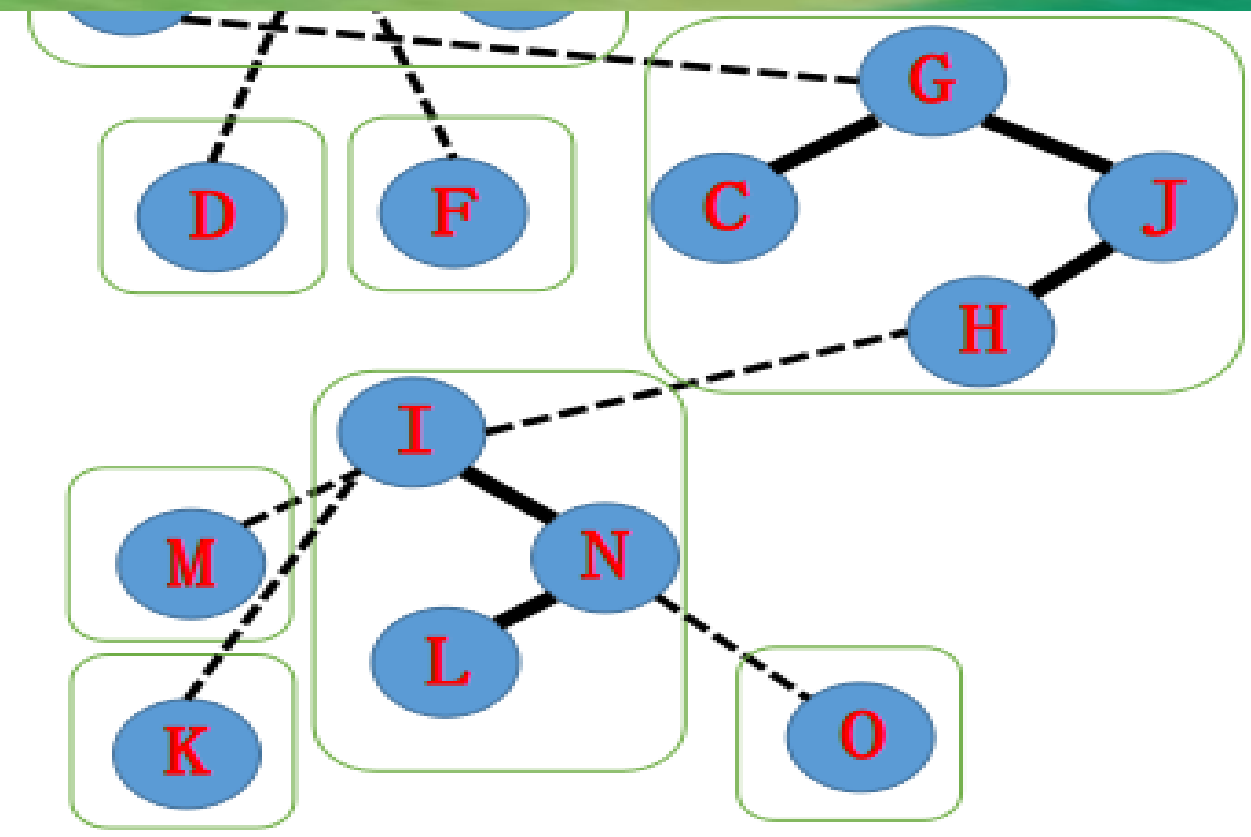
LCT核心操作:access

我们接着把N所属Splay的虚边指向的I（在原树上是L的父亲）也转到它所属Splay的根，splay(I)。

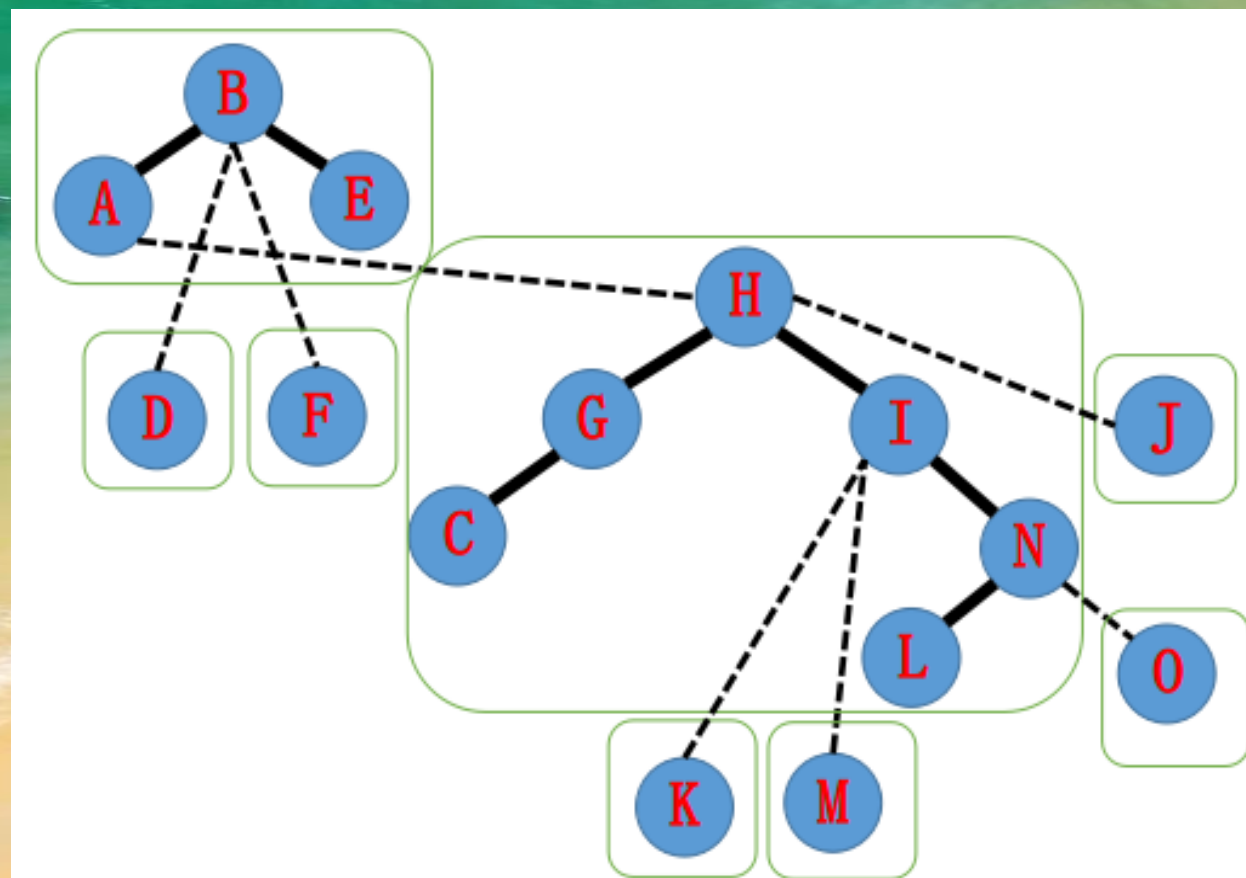
原来在I下方的重边I-K要变轻（同样是将右儿子去掉）。

这时候I-L就可以变重了。因为L肯定是在I下方的（刚才L所属Splay指向了I），所以I的右儿子置为N，满足性质1。

然后就变成了这样——

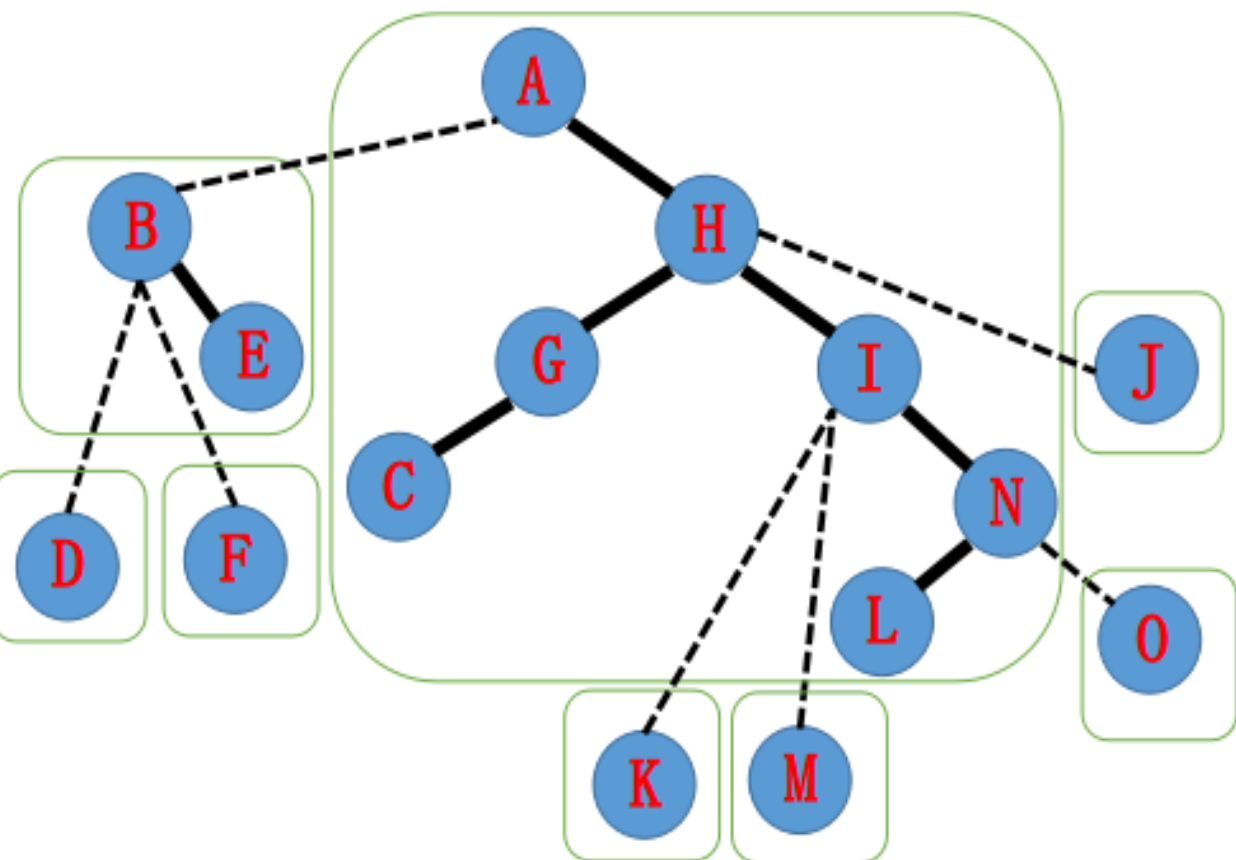


I指向H，接着splay(H)，
H的右儿子置为I。



LCT核心操作:access

H 指向A, 接着 $\text{splay}(A)$, A的右儿子置为H。A-N 的路径已经在在一个Splay中了, 大功告成!



- 代码其实很简单……循环处理, 只有四步——
- 1) 转到根;
- 2) 换儿子;
- 3) 更新信息;
- 4) 当前操作点切换为轻边所指的父亲, 转1

此处操作以模板题：洛谷 P3690 Link Cut Tree （动态树）为例

题意：维护四种操作

- 1) 询问从x到y的路径上的点的权值的xor和。保证x到y是联通的
- 2) 连接x到y，若x到y已经联通则无需连接
- 3) 删除边(x, y)，不保证边(x, y)存在
- 4) 将点x上的权值变成y

换根makeroot

只是把根到某个节点的路径拉起来并不能满足我们的需要。

更多时候, 我们要获取指定两个节点之间的路径信息。

然而一定会出现路径不能满足按深度严格递增的要求的情况。根据性质1, 这样的路径不能在一个Splay中。

访问access

access(x) 后x在Splay中一定是深度最大的点。

splay(x)后, x在Splay中将没有右子树（性质1）。于是翻转整个Splay, 使得所有点的深度都倒过来了, x没了左子树, 反倒成了深度最小的点（根节点），达到了我们换根的目的。

- pushup(int x) { //上传信息
- tag[x]=tag[son[x][0]]^tag[son[x][1]]^a[x];}
- access(int x) {
- for(int y=0;x=father[y=x])
- splay(x); //只传了一个参数,
- 因为所有操作的目标都是该Splay的根
- son[x][1]=y, pushup(x);}
- pushr(int x) //rev[]为翻转标记
- {swap(son[x][0], son[x][1]);
- rev[x]^=1;} //翻转操作
- makeroot(int x) {
- access(x); splay(x);
- pushr(x);}

部分操作：寻根findroot，分裂split

findroot:找x所在原树的树根，主要用来判断两点之间的连通性

(findroot(x)==findroot(y)表明x, y在同一棵树中)

- int findroot(int x) {
- access(x); splay(x);
- while(son[x][0])
- pushdown(x), x=son[x][0];
- //如要获得正确的原树树根,
- 一定pushdown!
- //同样利用性质1, 不停找左儿子, 因为其深度一定比当前点深度小。
- splay(x); //保证复杂度
- return x; }



split(x, y) 定义为拉出x-y的路径成为一个Splay (这里以y作为该Splay的根)

- void split(int x, int y) {
- makeroot(x);
- access(y); splay(y); }

- //x成为了根, 那么x到y的路径就可以用access(y)直接拉出来了, 将y转到Splay根后, 我们就可以直接通过访问y来获取该路径的有关信息

将x-y的边断开。

如果题目保证断边合法，倒是很方便。

使x为根后，y的父亲一定指向x，深度相差一定是1。

当access(y), splay(y)以后，x一定是y的左儿子，直接双向断开连接

正确姿势——先判一下连通性（注意findroot(y)以后x成了根），再看看x,y是否有父子关系，还要看y是否有左儿子。

因为access(y)以后，假如y与x在同一Splay中而没有直接连边，那么这条路径上就一定会有其它点，

在中序遍历序列中的位置会介于x与y之间。

那么可能y的父亲就不是x了。

连一条x-y的边（本蒟蒻使x的父亲指向y，连一条轻边）

- `bool link(int x, int y) {`
- `makeroot(x);`
- `if(findroot(y)==x) return 0;`
- //两点已经在同一子树中, 再连边不合法
- `father[x]=y; return 1;}`

• 如果题目保证连边合法，代码就可以更简单

- `void link(int x, int y) {`
- `makeroot(x); father[x]=y;}`

也可能y的父亲还是x，那么其它的点就在y的左子树中只有三个条件都满足，才可以断掉。

保证断边合法：

- `void cut(int x, int y) {`
- `split(x, y);`
- `father[x]=son[y][0]=0;`
- `pushup(y); //少了个儿子，也要上传一下}`

• 不保证断边合法：

- `bool cut(int x, int y) {`
- `makeroot(x);`
- `if(findroot(y)!=x || father[y]!=x || son[y][0])`
- `return 0;`
- `father[y]=son[x][1]=0; //x在findroot(y)后被转到根`
- `pushup(x); return 1;}`

补充：如果维护了size，还可以换一种判断

```
• bool cut(int x, int y) { //判断节点是否为一个Splay的根:isroot
•     makeroot(x);
•     if(findroot(y) != x || size[x] > 2)
•         return 0;
•     father[y] = son[x][1] = 0;
•     pushup(x); return 1; }
```

```
bool isroot(int x) { return
son[father[x]][0] == x ||
son[father[x]][1] == x; }
```

- //解释一下，如果他们直接连边的话，access(y)以后，为了满足性质1，该Splay只会剩下x, y两个点了。
 - 反过来说，如果有其它的点，size不就大于2了么？
- //原理很简单，如果连的是轻边，他的父亲的儿子们里没有它

题目环节！

内容由浅入深，难易交错，欢迎同学们踊跃抢答！

提醒：以下题目若无特殊说明
测评环境均视为jz oj测评机！

「BZOJ2002」[HNOI2010] Bounce 弹飞绵羊

- 题目：某天，Lostmonkey发明了一种超级弹力装置，为了在他的绵羊朋友面前显摆，他邀请小绵羊一起玩个游戏。游戏一开始，Lostmonkey在地上沿着一条直线摆上 n 个装置，每个装置设定初始弹力系数 $k[i]$ ，当绵羊达到第 i 个装置时，它会往后弹 $k[i]$ 步，达到第 $i+k[i]$ 个装置，若不存在第 $i+k[i]$ 个装置，则绵羊被弹飞。绵羊想知道当它从第 i 个装置起步时，被弹几次后会被弹飞。为了使游戏更有趣，Lostmonkey可以修改某个弹力装置的弹力系数，任何时候弹力系数均为正整数。
- 输入：第一行包含一个整数 n ，表示地上有 n 个装置，装置的编号从0到 $n-1$ ，接下来一行有 n 个正整数，依次为那 n 个装置的初始弹力系数。第三行有一个正整数 m ，接下来 m 行每行至少有两个数 i, j ，若 $i=1$ ，你要输出从 j 出发被弹几次后被弹飞，若 $i=2$ 则还会再输入一个正整数 k ，表示第 j 个弹力装置的系数被修改成 k 。
- 对于20%的数据 $n, m \leq 10000$ ，对于100%的数据 $n \leq 200000, m \leq 100000$
- 输出：对于每个 $i=1$ 的情况，你都要输出一个需要的步数，占一行。

样例

- Sample Input

- 4

- 1 2 1 1

- 3

- 1 1

- 2 1 1

- 1 1

- Sample Output

- 2

- 3

测评环境: bzoj评测机

Time Limits: 10 second

Memory Limits: 259 MB

题解：

法一：

这题只需要维护size就好了。

我们可以看到, 假如说对于 x , 它当前的弹力系数是 $a[x]$, 那么它会弹到 $x+a[x]$; 我们可以把 $x+a[x]$ 看作它的父亲。如此我们就可以建立起一棵树了。

查询事实上就是查询点 p 到根节点的距离。

至于修改, 我们看到,

这题的修改操作是修改一个点的父亲, 导致了树的结构发生了改变。

所以我们这个时候就要用LCT了: 用splay来维护信息

最后答案是 $size[left[v]]$, $left[v]$ 是点 v 的左儿子

法二：

顺便简略一提: 本题可以用分块做

分块思想很巧妙……

每个点记录跳出分块的步数以及跳到下一分块的哪个点……

jzoj3625 【SDOI2014】旅行(travel)

S 国有 N 个城市，编号从 1 到 N 。城市间用 $N - 1$ 条双向道路连接，满足从一个城市出发可以到达其它所有城市。每个城市信仰不同的宗教，如飞天面条神教、隐形独角兽教、绝地教都是常见的信仰。为了方便，我们用不同的正整数代表各种宗教。

S 国的居民常常旅行。旅行时他们总会走最短路，并且为了避免麻烦，只在信仰和他们相同的城市留宿。当然旅程的终点也是信仰与他相同的城市。S 国政府为每个城市标定了不同的旅行评级，旅行者们常会记下途中（包括起点和终点）留宿过的城市的评级总和或最大值。

Time Limits: 2000 ms

在 S 国的历史上常会发生以下几种事件：Memory Limits: 524288 KB

- "CC $x\ c$ ": 城市 x 的居民全体改信了 c 教；
- "CW $x\ w$ ": 城市 x 的评级调整为 w ；
- "QS $x\ y$ ": 一位旅行者从城市 x 出发，到城市 y ，并记下了途中留宿过的城市的评级总和；
- "QM $x\ y$ ": 一位旅行者从城市 x 出发，到城市 y ，并记下了途中留宿过的城市的评级最大值。

由于年代久远，旅行者记下的数字已经遗失了，但记录开始之前每座城市的信仰与评级，还有事件记录本身是完好的。请根据这些信息，还原旅行者记下的数字。

为了方便，我们认为事件之间的间隔足够长，以致在任意一次旅行中，所有城市的评级和信仰保持不变。

输入的第一行包含整数 N, Q ，依次表示城市数和事件数。

接下来 N 行，第 $i + 1$ 行两个整数 W_i, C_i 依次表示记录开始之前，城市 i 的评级和信仰。

接下来 $N - 1$ 行每行两个整数 x, y 表示一条双向道路。

接下来 Q 行，每行一个操作，格式如上所述。

对每个 QS 和 QM 事件，输出一行，表示旅行者记下的数字。

下表中 C 表示宗教总数。

测试点	N, Q	C	其它约定
1,2	$N, Q \leq 10^3$	$C \leq 10^2$	无
3,4	$N, Q \leq 10^5$	$C \leq 10^2$	S 国的交通网是一条链；无 CC 操作
5	$N, Q \leq 10^5$	$C \leq 10^2$	无 CC，QM 操作
6,7	$N, Q \leq 10^5$	$C \leq 10^2$	无 CC 操作
8,9	$N, Q \leq 10^5$	$C \leq 10^5$	S 国的交通网是一条链
10~12	$N, Q \leq 10^5$	$C \leq 10$	无
13~16	$N, Q \leq 10^5$	$C \leq 10^5$	无 QM 操作
17~20	$N, Q \leq 10^5$	$C \leq 10^5$	无

数据保证对所有 QS 和 QM 事件，起点和终点城市的信仰相同；在任意时刻，城市的评级总是不大于 10^4 的正整数，且宗教值不大于 C 。

题解：

- 所有点颜色相同时，问题可以用轻重边剖分或 Link-Cut Tree 解决。
- $C \leq 10$ 时，维护 C 棵树；第 i 棵树中，颜色为 i 的点的权值为 W_i ，其它点权为 0。颜色修改对应于权值修改。
- 链的情况下操作可以用线段树完成，可以用 C 棵动态存储的线段树解决。
- 无 QM 操作时，所有操作都对应于 dfs 序列上的线段树操作，同样可以用动态线段树解决。

- 无 CC 操作时，由于每棵树上的很多节点权值恒为零，我们可以
- 在第 i 棵树上只留下颜色为 i 的点，之后用链剖或 LCT 维护。
- 为了让留下的点能连边成树，我们还需要留下它们的一些 LCA。
- 可以发现，将所有点按 dfs 序排序之后，只留下相邻点的 LCA
- 就可以了。
- 留下的点数之和是 $O(N)$ 的，算法复杂度 $O(N \log N)$ 。
- 有 CC 操作时，对每种颜色，我们在对应的树上留下所有曾经为
- 这种颜色的点及相关的 LCA。可以发现留下的点数之和为
- $O(N + Q)$ ，算法复杂度 $O((N + Q) \log N)$ 。

提问：在线算法...？

jzoj100007 【SDOI2017】树点涂色

Bob有一棵 n 个点的有根树，其中1号点是根结点。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可能会进行以下几种操作：

- 1 x :

把点 x 到根结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，使得这个点到根结点的路径权值最大，求最大权值。

Bob一共会进行 m 次操作。

第一行两个数 n, m 。

接下来 $n - 1$ 行，每行两个数 a, b ，表示 a 与 b 之间有一条边。

接下来 m 行，表示操作，格式见题目描述。

每当出现2,3操作，输出一行。

如果是2操作，输出一个数表示路径的权值。

如果是3操作，输出一个数表示权值的最大值。

共10个测试点

测试点1, $1 \leq n, m \leq 1000$

测试点2、3，没有2操作

测试点4、5，没有3操作

测试点6，树的生成方式是，对于 $i (2 \leq i \leq n)$ ，在1到 $(i - 1)$ 中随机选一个点作为 i 的父结点。

测试点7, $1 \leq n, m \leq 50000$

测试点8, $1 \leq n \leq 50000$

测试点9、10，无特殊限制

对所有数据, $1 \leq n \leq 10^5, 1 \leq m \leq 10^5$

样例:

• Sample Input

• 5 6

• 1 2

• 2 3

• 3 4

• 3 5

• 2 4 5

• 3 3

• 1 4

• 2 4 5

• 1 5

• 2 4 5

Sample Output

3

4

2

2

Time Limits: 1000 ms

Memory Limits: 131072 KB

题解:

定义 $f(x)$ ，如果 x 与它的父节点颜色相同, $f(x)=1$, 否则 $f(x)=0$ 。认为 $f(1)=1$

定义 $g(x)$ 为 x 到1路径上所有点的 $f(x)$ 的和, $g(x)$ 就是1到 x 路径的权值。

1操作对一系列 $f(x)$ 修改，相应修改 $g(x)$ 。

2操作中， x 与 y 的lca为 z 则答案为 $g(x)+g(y)-2*g(z)+1$

3操作中，求自述中 $g(x)$ 的最大值

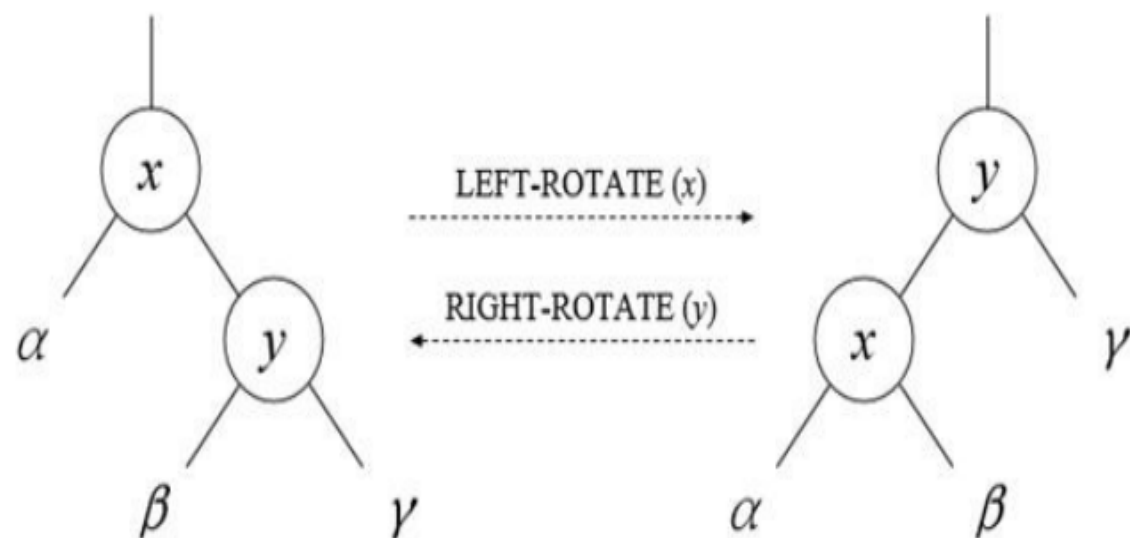
1操作就是LCT中的access操作, 用LCT可以快速得出哪些 $f(x)$ 发生改变。

2、3操作可以在树链剖分后用线段树维护信息。树链剖分时需要按dfs序编号，来支持3操作

jzoj5662 [GD0I2018Day1模拟4.17]尺树寸泓

这天小c在学习平衡树，却无法理解它神奇的旋转机制。特来请教她的好朋友小x。

小x随手掏出了一张图：



“看吧，这是一棵平衡树，如果对左图中的 x 进行左旋，就会变成右图，相应的，如果对右图中的 y 进行右旋，就会变回左图的样子。”

“唔……”小c若有所思。

“考你一个问题吧，你要维护一棵平衡树，定义一个节点的力量值是：子树内节点的权值和。我会让树进行旋转，并会随时询问你一个节点子树内力量值的积。”

小c思考片刻解决了这个问题，聪明的你会做吗？

从文件 `splay.in` 中读入数据。

输入文件的第1行两个数 n, Q ，表示点数和操作数。

后面 n 行，每行3个数， w_i, l_i, r_i ，表示这个点的权值、左儿子编号、右儿子编号。如果儿子=0表示没有这个儿子。保证这是一棵二叉树。保证初始时每个点儿子节点的编号（如果有的话）>这个点的编号。

后面 Q 行，每行2个数， opt, x 。

如果 $opt = 0$ ，表示右旋节点 x ，如果不合法（没有左儿子）请忽略这条操作。

如果 $opt = 1$ ，表示左旋节点 x ，如果不合法（没有右儿子）请忽略这条操作。

如果 $opt = 2$ ，表示询问节点 x 子树内所有点力量值的积。

输出到文件 `splay.out` 中。

输出文件对于每个操作2，输出1行表示这个询问的答案。对 $10^9 + 7$ 取模。

对于100%的数据，保证 $1 \leq n, Q \leq 2 \times 10^5, 0 \leq opt \leq 2, 1 \leq w_i < 10^9 + 7$ ，保证输出的不为0。

测试点编号	n	Q
1	$n \leq 10^3$	$Q \leq 10^3$
2		
3		
4	$n \leq 8 \times 10^4$	$Q \leq 8 \times 10^4$
5		
6		
7		
8	$n \leq 10^5$	$Q \leq 10^5$
9		
10	无	无

样例:

• Sample Input

• 3 4

• 1 2 3

• 1 0 0

• 1 0 0

• 2 1

• 0 1

• 2 2

• 2 1

Sample Output

3

6

2

Time Limits: 1000 ms

Memory Limits: 262144 KB

题解：

【标准算法 1】

形式化地描述要进行什么操作。

旋转节点：link/cut

更改两个点的子树和：单点修改。

询问一个点子树内子树和的积：子树询问。

可以在 LCT 上维护轻儿子信息，这样就可以子树询问了。轻重边切换的时候顺便维护这个。复杂度 $O(n \log n)$ 但是码量极大。

【标准算法 2】

子树询问的是积，具有可减性。

可以转化成link/cut链乘除一个值，单点询问。

用 LCT 维护。复杂度 $O(n \log n)$

题解：

【标准算法 3】

每次子树和只有 2 个点会修改。

问题在于 DFS 序会修改，导致子树信息比较麻烦。

注意到这是一棵平衡树，无论它怎么旋转，中序遍历是确定的。

而在中序遍历中，一个点的子树也是一个区间。

那么我们可以把中序遍历搞出来，然后维护每个点子树对应的区间。

一次旋转操作只有 2 个点的子树区间会更改。可以根据左旋还是右旋方便地讨论出来。

由于中序遍历序不变，只要用一个线段树维护就可以了。

然后就只剩下单点修改、区间求积了。复杂度 $O(n \log n)$

【问题来源】

2014 Multi-University Training Contest 7 《Game on Splay》

jzoj5387 [GD0I2018模拟9.23]动态图

众所周知，OI界的数据结构总是不断有创新的。

Axel正准备参加全国青少年数据结构奥林匹克竞赛，所以自然要充分学习多年以来的各种工业数据结构。于是Axel从线段树、Splay一直眼杀到了LCT、K-DTree、TopTree。接着他又做了几道动态仙人掌的题，然而一道题目将他卡住了~

Axel瞟了一眼题目，难道……这就是2015年某位大佬在某次讲课上的主题——动态图!!! Axel赶紧翻出那次的PPT，稍微学习了一下就把这道题A掉了。现在，他想让你来做一下这道题以检测他的工业基础是否雄厚。

给定一个 n 个点的图，初始时没有边。需要支持插入一条边，删除最后一条边，以及询问仅考虑第 l 到第 r 条边时的联通块数量。

第一行有三个整数 $n, q, type$ ， n 和 q 的含义见问题描述， $type$ 为强制在线参数。

接下来有 q 行，每行第一个整数为 t 。

当 $t = 1$ 时，接下来有两个整数 u, v ，解密后表示加入一条边 (u, v) ；当 $t = 2$ 时，表示删除最后加入一条边；当 $t = 3$ 时，接下来有两个整数 l, r ，解密后表示询问仅考虑第 l 到第 r 条边时联通块数量，注意这里边的编号是不考虑已删除的边的，即删除一条边后，再加入的边会使用之前删除的边的编号。

若 $type = 0$ ，则 u, v, l, r 解密后不变；若 $type = 1$ ，设 $lans$ 表示上一次操作3的答案（若之前没有操作3则为0）， m 表示当前边数，解密方法为： $u = (u \text{ xor } lans) \bmod n + 1, v = (v \text{ xor } lans) \bmod n + 1, l = (l \text{ xor } lans) \bmod m + 1, r = (r \text{ xor } lans) \bmod m + 1$ ，若解密后 $l > r$ ，则交换 l 和 r 的值。

对于每一个操作3，输出一个数，表示答案。

下表中的 $t_1 = 1, t_2 = 1, t_3 = 1$ ，分别对应数据中 $l = 1, r = m$ 、无操作2。

测试点	n	q	type	t_1	t_2	t_3
1	≤ 10000	≤ 20000	0	0	0	0
2			1			
3	≤ 1000000	≤ 200000	0	1	1	1
4					0	
5				0	1	1
6					0	0
7						
8						
9						
10						

对于100%的数据， $1 \leq n \leq 1000000, 1 \leq q \leq 200000, type \in \{0, 1\}, 1 \leq l, r \leq m$ ，可能有重边和自环。

样例:

Sample Input

5 3 0

1 4 5

1 2 4

3 1 2

Sample Output

3

Time Limits: 1000 ms

Memory Limits: 262144 KB

题解：（提问！离线算法？）

我们对于每一条边记录一个 $last[i]$ 。

如果这条边加入没有产生新环，则 $last[i] = 0$ 。

如果产生了新环，则 $last[i]$ 表示这个环上编号最小的边，并把这条编号最小的边删除。

当我们得到当前的 $last$ 数组以后，对于询问 $[l, r]$ 的答案就是 n 减去满足 $i \in [l, r], last[i] \in [0, l)$ 的 i 的数目。

证明也十分简单，因为一旦 $i \in [l, r], last[i] \in [0, l)$ ，就表明在只考虑 $[l, r]$ 的边时，加入的边 i 不会导致环，就使联通块数目减小了1。

$last$ 数组可以通过用LCT维护当前的边求得。

如何支持删除操作呢？因为删除只是删除最后一条边，所以我们可以用可持久化线段树来维护 $last$ 数组。并把 $last[m]$ 对应地加回LCT中。

嗯，口胡好题。

时间复杂度 $O(q \log(n + m) + q \log m)$ ，期望得分100pts。

jzoj5091 [GD0I2017第四轮模拟day2]绝版题

Description

所谓的考试，就一定有一道绝版题使得男人沉默女人流泪，而不有理有据的绝版题怎么称得上绝版呢？

火车国一开始只有一座城市，也就是1号城市。不过火车国的领土是在不断变化的，经常会新添加一个城市，那么小火车就会用一条铁路把它和某个老城市连接起来。

偶尔火车国会发生自然灾害，那么小火车就得找到一个合适的城市指挥赈灾，这个城市满足所有城市到其距离乘以城市人口的和最小，如果有不止一个最小的城市时小火车会选择距离1号城市最近的。

当然火车国人口也是不断变化的，也就是说有时候某个城市的人口会改变。现在小火车请你告诉他每次指挥赈灾应该选择的城市。

输入输出：

Input

第一行两个整数 m 和 t_1 ，表示事件数量以及1号城市初始人口。

接下来 m 行每行先是一个整数 $type$ 表示事件种类。

如果 $type=1$ 表示新建一个城市，编号为当前城市最大编号加一，接下来读入两个整数 u 和 t 表示新建一条连接着城市 u 和新城市的铁路，新城市的人口为 t 。

如果 $type=2$ 表示一个城市的人口发生了变化，读入两个整数 u 和 t 表示城市 u 的人口变成了 t 。

否则 $type$ 一定为3，表示一次询问。

为了体现问题的在线性，小火车对输入顺序进行了加密，用 $lastans$ 表示上一次的回答（初始为0），则读入的 u 和 t 都需要按位异或 $lastans$ 得到真正的操作。

Output

对于每个询问输出一行一个整数表示答案。

输入输出+数据范围:

Time Limits: 1000 ms

Memory Limits: 524288 KB

Sample Input

7 1

1 1 1

1 1 1

1 2 2

1 2 1

3

2 1 1

3

Sample Output

2

1

Data Constraint

对于20%的数据 $m \leq 5000$;

对于另外30%的数据

保证最后一个1操作之前没有询问;

对于另外30%的数据保证没有2操作;

对于100%的数据 $m \leq 300000$, 其中1操作和2操作个数不超过150000, 保证任意城市人口数量不超过1000000且不低于1。

题解：

可以证明答案一定为树的带权重心，也就是最大子树最小的节点，实际上也就是将某个点作为根之后深度最大的子树大小超过一半的节点，不妨用`lct`维护每个点的子树大小，现在考虑如何求出答案。

不妨从根所在链自上而下搜索节点，先在当前的链上二分出最深的满足条件的点，再走向其最大的虚儿子看是否可行，最后再把答案`access`一下，可以发现这样的搜索节点过程复杂度与`access`相同。

当然每个节点需要开一个`set`维护虚儿子大小。

时间复杂度： $O(n(\log n)^2)$

jzoj3609 [NOI2014模拟]重组病毒

黑客们通过对已有的病毒反编译，将许多不同的病毒重组，并重新编译出了新型的重组病毒。这种病毒的繁殖和变异能力极强。为了阻止这种病毒传播，某安全机构策划了一次实验，来研究这种病毒。

实验在一个封闭的局域网内进行。局域网内有 n 台计算机，编号为 $1 \sim n$ 。一些计算机之间通过网线直接相连，形成树形的结构。局域网中有一台特殊的计算机，称之为核心计算机。根据一些初步的研究，研究员们拟定了一个一共 m 步的实验。实验开始之前，核心计算机的编号为 1 ，每台计算机中都有病毒的一个变种，而且每台计算机中的变种都不相同。实验中的每一步会是下面中的一种操作：

1、RELEASE x

在编号为 x 的计算机中植入病毒的一个新变种。这个变种在植入之前不存在于局域网中。

2、RECENTER x

将核心计算机改为编号为 x 的计算机。但是这个操作会导致原来核心计算机中的病毒产生新变种，并感染过来。换言之，假设操作前的核心计算机编号为 y ，相当于在操作后附加了一次RELEASE y 的操作。

根据研究的结论，在植入一个新变种时，病毒会在局域网中搜索核心计算机的位置，并沿着网络中最短的路径感染过去。

而第一轮实验揭露了一个惊人的真相：病毒的不同变种是互斥的。新变种在感染一台已经被旧变种感染的电脑时，会把旧变种完全销毁之后再感染。但研究员发现了实现过程中的漏洞。如果新变种在感染过程中尚未销毁过这类旧变种，需要先花费 1 单位时间分析旧变种，才能销毁。如果之前销毁过这类旧变种，就可以认为销毁不花费时间。病毒在两台计算机之间的传播亦可认为不花费时间。

研究员对整个感染过程的耗时特别感兴趣，因为这是消灭病毒的最好时机。于是在步实验之中，研究员有时还会做出如下的询问：

3、REQUEST x

询问如果在编号为 x 的计算机的关键集合中的计算机中植入一个新变种，平均感染时间为多长。编号为 y 的计算机在编号为 x 的计算机的关键集合中，当且仅当从 y 沿网络中的最短路径感染到核心计算机必须经过 x 。由于有RECENTER操作的存在，这个集合并不一定是始终不变的。

至此，安全机构认为已经不需要实际的实验了，于是他们拜托你编写一个程序，模拟实验的结果，并回答所有的询问。

Time Limits: 2000 ms Memory Limits: 524288 KB

Input

输入的第一行包含两个整数 n 和 m ，分别代表局域网中计算机的数量，以及操作和询问的总数。

接下来 $n-1$ 行，每行包含两个整数 x 和 y ，表示局域网中编号为 x 和 y 的计算机之间有网线直接相连。

接下来 m 行，每行包含一个操作或者询问，格式如问题描述中所述。

Output

对于每个询问，输出一个实数，代表平均感染时间。输出与答案的绝对误差不超过 10^{-6} 时才会被视为正确。

所有测试点的数据规模如下：

测试点编号	n	m	备注
1	$n \leq 100$	$m \leq 100$	无
2			
3	$n \leq 100000$	$m \leq 100000$	网络呈一条链 且不存在 RECENTER 操作
4			网络呈一条链
5			
6			
7			
8			
9	$n \leq 30000$	$m \leq 30000$	操作和询问的计算机关键 集合的大小均为 1 不存在 RECENTER 操作
10			
11	$n \leq 50000$	$m \leq 50000$	不存在 RECENTER 操作
12			
13			
14			
15	$n \leq 100000$	$m \leq 100000$	无
16			
17			
18			
19			
20			

Sample Input

8 6

1 2

1 3

2 8

3 4

3 5

3 6

4 7

REQUEST 7

RELEASE 3

REQUEST 3

RECENTER 5

RELEASE 2

REQUEST 1

Sample Output

4. 0000000000

2. 0000000000

1. 3333333333

题解：

3 重组病毒 Recompile

3.1 简述

给定一棵 n 个点的有根树，初始时每个节点都有一个不同的颜色。定义一个节点的代价为其走到根遇到的不同颜色种数。有 m 次操作或询问：

1. 将某个点到根的路径上的所有点的颜色改为一种新的颜色。
2. 将树根改为某个节点，同时将两个根之间的路径上所有点改为一种新颜色。
3. 查询某个节点子树里所有节点代价的平均数。

$n, m \leq 100000$ 。

3.2 分析

3.2.1 10%的数据

n 和 m 都特别小，随便写个暴力就可以拿到分了。如果暴力写的比较好，在后面还可能拿到更多分数。

3.2.2 初步分析

由于题目特殊的修改方式，不难得出，在任意一条到根的路径上，相同的颜色的节点必然是连续一段。这样我们可以重新定义代价：把代价转到边上，如果一条边连接的两个节点颜色不同，则权值为1，否则为0。那么一个点的代价 $c(x)$ 就是从 x 到根的边权和+1。

3.2.3 链的数据

有了上面的分析，这部分数据应该可以用线段树之类的数据结构直接做。

3.2.4 不存在换根的数据

从这里开始就接近正解了。我们继续观察，发现这棵树和我们常用的某种数据结构很相似—Link-Cut Tree。权值为1的边就是虚边，为0的边就是实边。初始时所有边都是虚边，代价 $c(x)$ 为从 x 到根遇到的虚边条数+1。

那么操作呢？操作1实际上就是expose（也称access）一个点。由于LCT的总复杂度为 $O(n \log n)$ ，可知操作的边数为 $O(n \log n)$ 级别的。那么我们实现一棵LCT，就可以在expose的过程中找到需要改为实边的虚边，和需要改为虚边的实边。只要对于每个点维护其子树中最左侧的节点即可。

而剩下的就不难维护了。修改一条边相当于对一棵子树的所有节点的代价+1或-1，因此用线段树维护DFS序即可。整个算法的复杂度为 $O(n \log^2 n)$ ，但实际中达不到这个复杂度。

3.2.5 换根操作

题中的换根也比较特殊，特殊在于它附加了一些奇怪的操作。LCT是支持换根的，具体实现是expose x ，splay x ，再给 x 打上翻转标记。而题目中的奇怪操作恰好就对应了换根中的expose。那么我们对每个节点维护翻转标记（以及子树中最右侧的节点，为了快速打标记），LCT方面就可以对付了。

比较棘手的是线段树方面。但是我们可以研究DFS序的性质。假设新的根为 $root$ ，查询节点为 x ，在原树中 x 和 $root$ 只有三种关系：

1. $x = root$ 。查询整棵树。
2. x 在 $root$ 的子树内，或者 x 和 $root$ 不存在包含关系。查询原树中 x 对应的区间即可。
3. $root$ 在 x 的子树内。设 p 为 x 的所有儿子中是 $root$ 的祖先的那个儿子，那么此时 x 的子树应为原树中 x 子树外的所有节点、 x 本身，以及 x 非 p 的所有儿子的子树。换句话说，就是整棵树除去 p 的子树，而这样最多对应两个区间。 p 可以用倍增在 $O(\log n)$ 的时间内求出。

至此我们得到了正解的算法。复杂度为 $O(n \log^2 n)$ ，可以得到满分。

3.3 来源

不存在换根的部分来自于CodeChef November Challenge 2013 Gangsters of Treeland²。而用线段树维护换根后的子树和的思路来自于Tsinsen A1353 - 树（罗雨屏）³。

jzoj4427 [HNOI2016模拟4.4]Alphadog

3.1 题目描述

2020年，一堆废铁渣中，阿尔法狗苟延残喘。他万万没有想到可耻的人类居然欺骗了他。依然记得，4年前，他在棋局上将李世石九段打败，但那竟是韩国人的阴谋。韩国人是不会让人工智能发展起来的！韩国人让阿尔法狗自信满满，全世界地挑战棋手，然后乘机将阿尔法狗的套路学习得行云流水，最终阿尔法狗倒在了韩国人的手中！但作为人工智能中的佼佼者，阿尔法狗岂能容忍这种行为？

于是他开始研究人类基因中的特殊点，从而找到人类的弱点！他现在找到了一条DNA链，但上面有些碱基他暂时无法识别，假如他识别出了一个碱基，他会立即把他补充到DNA链上。对于一条DNA链 S 的特殊值 $F(S)$ 定义为

$$F_S = \sum_{1 \leq x \leq y \leq |S|} \text{LCP}(x, y)$$

LCP即Longest Common Palindrome的缩写，也就是说 $\text{LCP}(x, y)$ 表示最长的字符串 T 的长度，其中 T 要满足三个条件：

- T 是回文的
- 存在一个 $i \leq x$ ，满足 $S_{i..x} = T$ ， $S_{i..j}$ 表示 S 中从第 i 位到第 j 位的子串。
- 存在一个 $j \leq y$ ，满足 $S_{j..y} = T$

但阿尔法狗的CPU散热不强，他很怕自己会爆炸。于是他找到了你，忠实的AI支持者，为他服务。

第一行包含两个正整数 N, sig ， N 表示阿尔法狗得到的DNA链的长度。

接下来 N 行，第 i 行包括一个整数 x ，当 $sig = 1$ 时， x 要异或 $lstans$ 才是真正的 x ，否则若 $sig = 0$ ，则不需要异或。 $lstans$ 表示 $F(S_{1..i-1})$ 。 x 表示 $S_i = x$ 。

字符串的位置从1开始编号。初始时 $lstans = 0$ 。

输出包括 N 行，第 i 行表示 $F(S_{1..i})$ 。

对于20%的数据： $N \leq 100$

对于40%的数据： $N \leq 5000$

另外有20%的数据： $sig = 0$

对于100%的数据： $N \leq 10^5, 0 \leq x < 2^{63}, sig \in \{0, 1\}$

数据保证所有真正的 x 满足 $0 \leq x \leq 25$

Time Limits: 1000 ms

Memory Limits: 524288 KB

样例+解释:

Sample Input1:

```
6 1
0
0
3
5
12
14
```

Sample Input2:

```
6 0
1
1
1
1
1
0
0
```

Sample Output1:

```
1
2
5
10
11
12
```

Sample Output2:

```
1
4
10
20
21
24
```

3.5 样例解释

假如我们用小写字母来表示 S_i ，那么样例一中真正的 S 为 $abbagf$ ，因为样例二不需要异或，所以真正的 S 为 $bbbbaa$ 。

题解：

20~30pts:

除了manacher外，还有一种更为强力的字符串回文子串处理工具——回文树 Palindromic-Tree。这种妙不可言的数据结构能够匹配当前字符串的每一个回文子串。

由于顺着失配边能够从长到短枚举当前位置的每一个回文后缀，所以两个前缀的LCP就是他们在PT中对应的节点在失配树上的LCA。那么利用这个性质，我们大概能拿到二三十的暴力分。

40pts:

不妨考虑当前最长回文后缀对应节点p的每一个祖先对答案的贡献，那么得到下式：

$\Delta ans = \sum_{y \text{ 为 } p \text{ 的祖先}} len(y) * (size(y) - size(x))$, x为y的儿子且x为p的祖先
至此，维护size，我们可拿到四十分。

题解：（满分算法）

接下来该上大杀器——LCT了。转换上式可得：

$$\Delta \text{ans} = \sum (\text{y为p的祖先}) \text{size}(\text{y}) * (\text{len}(\text{y}) - \text{len}(\text{father}(\text{y})))$$

由于乘号右边是一个常量，而左边每次只有一条链被修改，显然 $\text{access}(\text{p}) + \text{lazy_tag}$ 就可以 $\log n$ 维护了。

注：此方法来自博客：

<https://blog.csdn.net/jazengm/article/details/75194439>

该博主：多么妙的一道LPT (Link-Palindromic-Tree) 啊！

jzoj4753 [GD0I2017模拟9.4]种树

环环很喜欢种树，隔壁的健健发现最近环环不知道从哪里搞来了一棵奇怪的树。这棵树刚种下时每个点都有自己**独特**的颜色。与一般的树一样，任何时刻它都**只有一个根**（初始根节点为1号点）。“但是环环这么强，她种的树怎么可能只会用五颜六色的外表来卖萌呢？”一直在暗中观察的健健想。于是健健又偷偷观察了几天，终于看到了一些神奇的事情：

这棵树的某个点有时会 and 根进行“交流”，它到根的路径上所有节点的颜色都会渐渐统一为一种**从未出现过的**颜色。

更神奇的是，这棵树有时会翻个身，它会拔起根，然后把另外的一个点插到泥土里使它变成新的根，然后新的根会和原来的根进行一次“**交流**”。

正当健健以为发现环环之所以强的秘密时，不幸的事发生了。环环碰到了正在偷偷观察的健健，环环便说：“你来的正好，我正想考你一个问题，现在这个点的子树中所有点到根的路径上平均有多少种颜色呢？昨天那个点的子树中所有点到根的路径上平均又有多少种颜色呢……”环环一连串的问题把健健问晕了，为了不让健健再被环环鄙视，健健请你帮他回答这些问题。如果你不能帮健健正确回答完这些问题，环环就会(*哔哔*)健健了。为了健健的人生大事，请你写一个程序帮助他。

第一行有两个正整数 n 和 m ，分别代表这棵树的节点数和健健遇到的事件数。

接下来 $n-1$ 行，每行有两个正整数 a, b ，表示节点 a 和 b 之间有树枝连接。

接下来 m 行，表示健健遇到的事件：

- (1) `Make_Root x` 表示这棵树翻了个身，新的根为节点 x ，同时原来的根会与现在的根进行一次交流，字符串与整数之间有一个空格；
- (2) `Paint x` 表示节点 x 与根进行了一次交流；
- (3) `Query x` 表示环环向健健问了一个问题，询问 x 的子树中所有节点到根的路径上平均有多少种颜色。

对于每个询问，输出一个实数（你的答案被视为正确，当且仅当与标准答案的绝对误差不超过 $1e-6$ ）

表示以 x 为根的子树中，所有节点到根的路径上的平均颜色数。

Time Limits: 2000 ms

Memory Limits: 262144 KB

题目大意：

给定一颗N个节点的有根树，初始时每个叶子节点都有一个不一样的颜色。定义一个节点的代价为其走到根遇到的不同颜色种数。有M次询问，共有三种类型：

1. 将节点u到根的路径上的所有点的颜色改成一种新的颜色。
2. 将树根改为节点u，同时将两个根之间的路径上所有点改为一种颜色。
3. 查询节点u子树里所有节点代价的平均数。

$N, M \leq 10^6$

Time Limits: 2000 ms

Memory Limits: 262144 KB

输入输出+数据范围:

Sample Input

8 6
1 2
1 3
2 8
3 4
3 5
3 6
4 7

Query 7

Paint 3

Query 3

Make_Root 5

Paint 2

Query 1

Sample Output

4.0000000
2.0000000
1.3333333

对于 10%的数据, $n \leq 100, m \leq 100$;

对于另外 15%的数据: $n \leq 100000, m \leq 100000$, 树呈一条链, 且不存在 Make_Root 操作;

对于另外 15%的数据: $n \leq 100000, m \leq 100000$, 树呈一条链;

对于另外 10%的数据: $n \leq 30000, m \leq 30000$, 且不存在 Make_Root 操作;

对于另外 20%的数据: $n \leq 50000, m \leq 50000$, 且不存在 Make_Root 操作;

对于 100%的数据: $1 \leq n \leq 100000, 1 \leq m \leq 100000$ 。

题解：

我们先考虑一个子问题，假设没有第二个操作，我们应该怎么维护？

如果没有第二个操作，那么树的根是确定的，也就是说树的DFS序是确定的。我们发现对于第一种操作很像LCT中的Access操作。所以我们考虑用LCT来维护。



那么模型就可以转化为：虚边的权值为1，实边的权值为0，初始时全部变都是实边，一个节点 u 的代价就是到遇到的虚边数+1。那么每次Access操作我们就需要把一些虚边改成实边，把一些实边改成虚边。

题解：

1. 对于把虚边改成实边：在Access时假设当前已合并的平衡树中最上端的点的父亲为 v ，如果 v 的下方有节点也处于 v 所在的平衡树中，那么这棵平衡树内就要把一条实边转成虚边，也就是把 v 下端，最上方的点打上+1的tag。

2. 对于把实边改成虚边：就是Access每次扩大平衡树时都会把一条虚边改成实边，那么只要把当前平衡树中最上端的点打上-1的tag。至于维护区间和，只要以DFS序为下标用线段树维护。

（这就是CC MONOPLOY）。

注：此方法来自博客：

<https://blog.csdn.net/YxuanwKeith/article/details/52474295>

题解：

那么我们考虑又换个操作的情况，如果有了换根操作，那么DFS序就会乱，线段树维护的值就会有问题。那么我们考虑不改变DFS序，只是在打tag的是后考虑一下在原树中根 $root$ 与当前结点 v 的关系。

1. $v=root$ ：查询整棵树。

2. v 在 $root$ 的子树内或 x 和 $root$ 不存在包含关系：直接查询原DFS区间。

3. $root$ 在 v 的子树内。设 p 为 v 的所有儿子中是 $root$ 祖先的那个儿子。那么此时 v 的子树应为原树中 v 的子树外所有的节点、 v 本身以及 v 中非 p 的所有儿子的子树。

那么每次查询和修改再加以判断一下就可以维护换根操作。

jzoj5157 [NOI2017模拟6.22]没有上司的舞会

小 **B** 供职的公司里，每天都会有新的员工加入。

每个新加入的员工，会有一个老司机作为他的直接上司。事实上，员工之间按照“直接上司”关系，形成了一个树的结构。

每天晚上，公司会举办一个舞会，大家可以自愿选择参加。但是，对于一个人来说，如果他的直接上司要参加舞会，那么他在会场就会很拘束，因此他会选择委婉拒绝参加这天的舞会而回家打 CodeForces。

最开始的时候，公司里只有编号为 **0** 的老总。在接下来的 n 天，每天都会有一个新员工加入公司；其中，第 i 天加入公司的人编号为 i ，其直接上司为 f_i 。

小 **B** 作为舞会负责人，为了准备物资，他需要在每天下午，统计当天晚上的舞会中最多可能会有多少人参加。但是随着公司的人越来越多，他有点算不过来了，于是他找到了你来帮忙。

为了体现这是一个实时性的任务，部分子任务采取了一些手段来进行强制在线。

从 `party.in` 读入数据。

输入的第一行包含一个整数 n ($1 \leq n \leq 3 \times 10^5$), $type$ ($type \in \{0, 1\}$)。分别表示你需要计算的天数和这组数据类型。

接下来 n 行，每行包含一个整数。其中第 i 行的数表示 \hat{f}_i 。你需要根据 \hat{f}_i 计算出 f_i ，公式如下：

$$f_i = \hat{f}_i \oplus (type \times \text{lastans})$$

其中， \oplus 表示 XOR 操作；**lastans** 表示上一天的答案，它的初始值为 0。

输出到 `party.out`。

输出 n 行，每行一个整数。其中第 i 行的数表示第 i 天的答案。

Time Limits: 1500 ms

Memory Limits: 262144 KB

Data Constraint

测试点	n	$type$	性质
1 ~ 4	≤ 3000	-	-
5 ~ 12	-	$= 0$	-
13 ~ 16	-	-	数据随机
17 ~ 20	$\leq 1.5 \times 10^5$	-	-
21 ~ 25	-	-	-

其中，“数据随机”的意义为：所有的 f_i 在 $[0, i)$ 内均匀等概率随机产生。

Sample Input

4 0
0
0
1
3

Sample Output

1
2
2
3

题解：

题目大意：维护一棵树，初始为空

- ▶ 支持动态加点、求最大独立集
- ▶ 强制在线
- ▶ $n \leq 2 * 10^5$

法一：考虑朴素DP

- ▶ 令 $f[i], g[i]$ 分别表示以 i 为根的子树，根节点选 / 不选的最优方案
- ▶ 真 • 没有上司的舞会 • 弱化版
- ▶ 单次询问复杂度 $O(n)$ ，总复杂度 $O(n^2)$

题解：

法二：数据随机，树的高度期望为 $O(\log n)$

- ▶ 不难注意到每次加点会影响到一条链上的 $f[]$ 和 $g[]$ ，暴力更新即可
- ▶ 复杂度期望 $O(n \log n)$

法三(离线情况)：允许离线

- ▶ 我们可以知道树的形态，从而对树进行树链剖分
- ▶ 接下来有若干种处理思路(见法三-1，法三-2)

题解:

法三-1: 考虑支持修改点权的带权最大独立集问题

- ▶ 考虑链上的情况:
- ▶ 建段树, 每个区间维护 $f[2][2]$ 表示左端点选 / 不选, 右端点选 / 不选的最优解
- ▶ 我们对每个重链维护一个这样的线段树, 遇到轻边则暴力跳父亲更新即可
- ▶ 单次修改复杂度 $O(\log^2 n)$
- ▶ 套到这道题上, 我们可以每次把一个点的权值由 0 修改为 1, 然后询问全局的带权最大独立集
- ▶ 总时间复杂度 $O(n \log^2 n)$

题解：

法三-2：树是天然的二分图，二分图上最大独立集 = 点数 - 最大匹配

- ▶ 考虑维护最大匹配
- ▶ 在树上的匹配具有贪心性质，我们每次选择尽量深的未匹配点及其父亲进行匹配
- ▶ 加入一个新叶子的时候，尝试向上寻找交错路，时刻维护上述性质，就不需要考虑“交错路拐弯”的情冢了
- ▶ 用线段树维护

题解：（满分算法）

考虑如何在线

- ▶ 我们可以使用 Link-Cut-Tree
- ▶ 上述两种算法都可以扩展到LCT上，核心思路为：

维护好重链的答案，暴力跳轻边的答案

- ▶ std 经过一些微小地推导，使用了一种常数更小的写法

未完待续……

(续) 一些推导:

我们给每个点定义一个“黑白”颜色:

- ▶ 首先叶子是黑色的
- ▶ 若一个点是黑色的, 则它的父亲是白色的
- ▶ 若一个点的所有孩子都是白色的, 则它本身是黑色的
- ▶ 答案即为黑色点数
- ▶ 从匹配或者独立集问题入手, 考虑问题在树上的贪心性质, 都

能发现上述结论

(再续) 实现细节:

显然地, 加入一个新的叶子会翻转一条链上的颜色

- ▶ 如果发现了这个结论, 但是没有写高级数据结构来维护颜色, 每次 $O(\text{树高})$ 地暴力找翻

转颜色的链顶, 能比最朴素地 $O(\text{树高})$ 暴力多获得 16 分

- ▶ 因为有 4 组数据是这样生成的:

- ▶ `for (int i = 1; i <= n; ++i)`

`addEdge(i - 1 - rand() % min(i, 50), i);`

- ▶ 考虑用 LCT 维护颜色, 我们时刻保持每条重链上都是黑白点交替出现

- ▶ 加入一个叶子后, 我们对其进行特殊的 `access`, 得到一条尽可能长的重链, 要求重链上

黑白点交替出现, 且每个黑点至多只有一个白点作为孩子

- ▶ 翻转整条链的颜色, 根据链顶的颜色, 考虑是否会令答案 +1

- ▶ 时间复杂度 $O(n \log n)$, 常数较小

(续续续) 扩展

这个问题可以推广为支持 link、cut、带点权和修改点权的完全动态形式

希望有缘人可以告诉我思考的结果！

古智锋和程子奇在这里祝诸君武运昌隆！