动态树分治

成都七中　nzhtl1477

常用的树分治方法

- 1. 点分治
- 2. 链分治
- 3. 边分治
- 4.link cut tree
- 5.top tree

Notice

- 目前我见过的类树分治的东西有:
- 点分治
- 边分治
- 链分治 --- 树链剖分 --- 树上启发式合并
- lct 维护子树
- 静态 lct 维护子树
- 类 top tree
- Top tree
- 静态 top tree

Note

- 其实会点分治 / 链分治就够用了
- Top tree 类的东西现在基本上没人会的，所以不用管
- 事实是 top tree 是目前我觉得的最通用化的树分治处理数据结构

静态树分治

- 每次找一个分治中心，计算跨过分治中心的所有答案，然后删去分治中心，继续递归分治下去计算
- 把所有分治中心的答案合并起来就是全局的答案

点分治

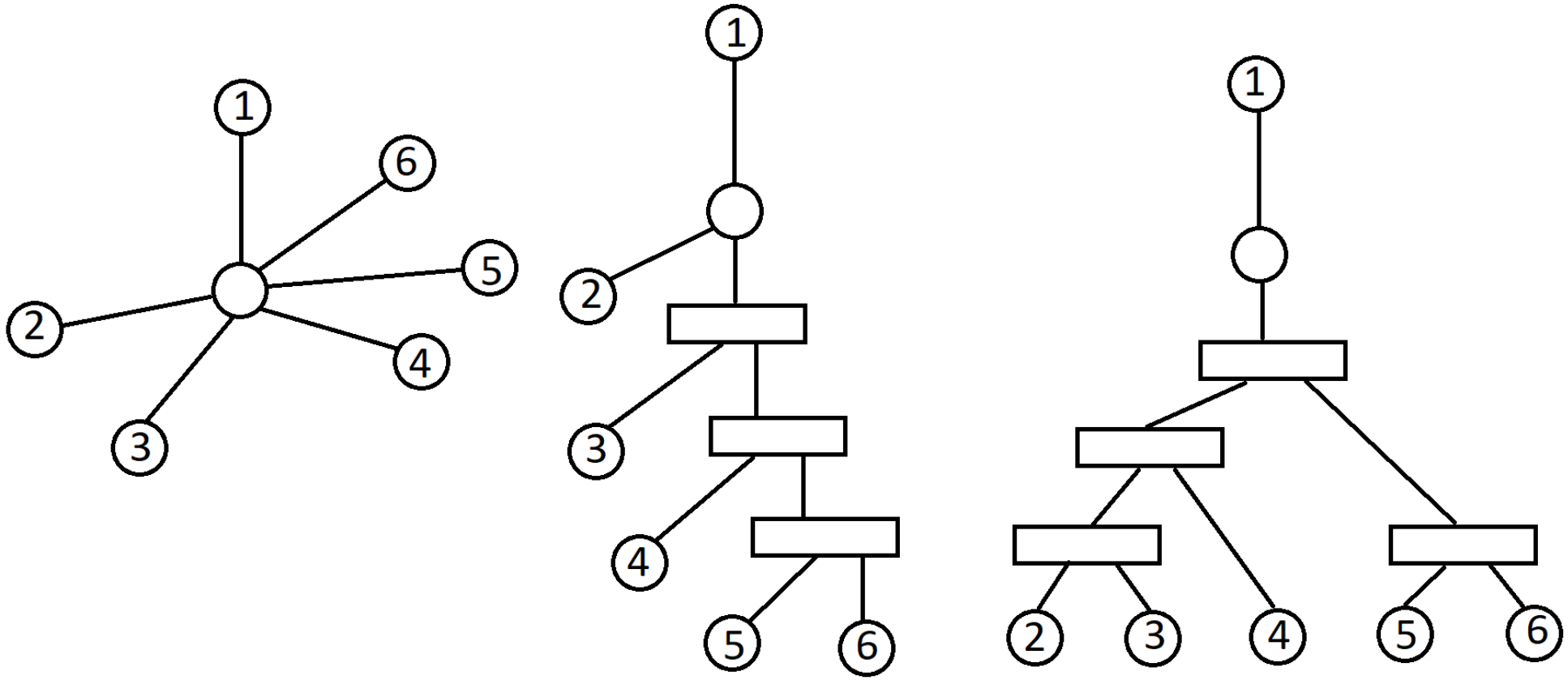
- 可以看作是每次删去一个点之后继续分治下去
- 每次找树的一个重心，然后递归每个子树
- 这些找到的重心构成一个树形结构
- 因为这个分治中心构成的树的深度 O(logn)
- 所以还是在每个分治中心维护一个数据结构
- 修改和查询都暴力跳就可以了

边分治

- 可以看作是每次删去一条边之后继续分治下去
- 删去一条边之后树会分成两个子树，删去的边需要保证让两边的子树 size 尽可能相近
- 直接朴素地在这个树上做可能复杂度不正确，需要进行三度化

三度化

- 对于每个度数 >3 的节点，可以通过加虚点的方法让该节点度数变为 3



三度化

- 注意三度化只是可以解决部分树点度数过大的问题，因为其本质并不是对点度数进行了分治，而仅仅是改变了树的结构

链分治

- 基于 HLD 的分治结构
- 可以看作是每次删去一条重链之后继续分治下去
- 每个重链头开一个数据结构
- 每次修改的时候跳重链，在每个重链头的数据结构上进行修改
- 查询同理

动态树分治

- 静态树分治每次选取的分治中心构成了一个树形结构
- 我们把这个树建出来，然后每次修改的时候可以在上面维护信息，这样就不用每次重新做一遍了
- 类比：
- 比如全局最大子段和可以分治做，单点修改全局最大子段和可以线段树来做，线段树就是把序列上的分治结构存下来了，每次在上面维护信息，从而避免了大量的重复计算

动态树分治

- 分治信息：只有 0(1) 的信息量，可以高效合并，有交换律，也就是一个半群
- 树链剖分：支持链 / 子树修改，查询链 / 子树分治信息
- 先看几个简单的树上问题?

Luogu4216 [SCOI2015] 情报传递

- 给你一棵树，初始每个位置没有点权
- 1 x：让一个点从当前时刻开始，每秒操作点权 ++
- 2 x y c：查询一条链中有多少点的点权大于 c
- 其中每秒操作点权 ++ 就是指我每操作一次，无论是否和那个点有关，那个点权值都会 ++
- 1 操作对于每个点只会开始一次

Solution

- 考虑我们肯定不能每次把 1 操作的点 ++ ，这样复杂度不对
- 看看 2 操作吧

Solution

- 查询 x -> y 中有多少点权 >=c 的点
- 而点权是一秒 +1
- 所以等价于查 x -> y 中有多少当前时间 - 开始时间 >=c 的点
- 即查询有多少点满足当前时间 -c>= 开始时间
- 相当于查一条链中小于一个数个数的数
- 这样可以树链剖分 + 平衡树做到 O(mlog^2n)

Solution2

- 进一步研究
- 如果第 i 次操作是 1 x，则相当于第 i 时刻 x 点开始
- 可以把操作 1 当成第 i 时刻的时候点 x 的权值从 0 变成了 1
- 如果第 i 次操作是 2 x y c，则相当于查询链 x -> y 中开始时间 <=i-c 的点的个数
- 等价于查询第 i-c 时刻链 x -> y 上的和
- 通过之前讲的树上差分 + 树状树组直接维护即可
- O(mlogn)

Luogu4211 [LNOI2014]LCA

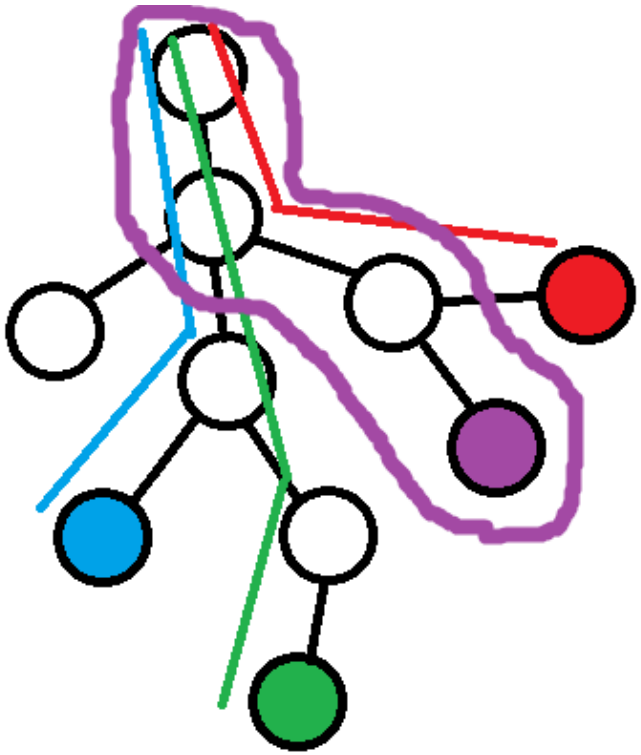
- 给出一个 n 个节点的有根树（编号为 0 到 n-1，根节点为 0）。
  一个点的深度定义为这个节点到根的距离 +1。
  设 dep[i] 表示点 i 的深度，LCA(i,j) 表示 i 与 j 的最近公共祖先。
  有 q 次询问，每次询问给出 l r z，求 sigma_{l<=i<=r}dep[LCA(i,z)]。
  （即，求在 [l,r] 区间内的每个节点 i 与 z 的最近公共祖先的深度之和）

Solution

- 首先将查询差分
- (l,r) -> (1,r) - (1,l-1)
- 然后考虑给一个点，怎么求其到一个前缀的点的 LCA 的深度和

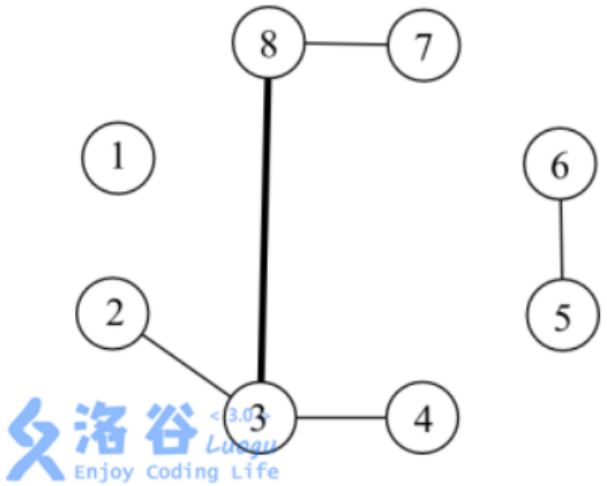
Solution

- 可以把每个点按顺序依次插入，每次插入把这个点到根的路径 ++ ，查询 z 到这些点各自的 lca 的深度和即查询 z 到根路径的和
- 如图，绿色，红色，蓝色的点被插入，
- 查询紫色的点
- O(mlog^2n)



Luogu4219 [BJOI2014] 大融合

小强要在N个孤立的星球上建立起一套通信系统。这套通信系统就是连接N个点的一个树。 这个树的边是一条一条添加上去的。在某个时刻，一条边的负载就是它所在的当前能够 联通的树上路过它的简单路径的数量。

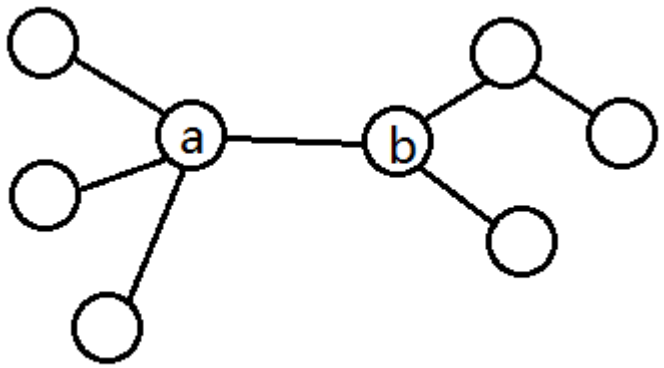


例如，在上图中，现在一共有了5条边。其中，$(3,8)$这条边的负载是6，因 为有六条简单路径$2-3-8$，$2-3-8-7$,$3-8$,$3-8-7$,$4-3-8$,$4-3-8-7$路过了$(3,8)$。

现在，你的任务就是随着边的添加，动态的回答小强对于某些边的负载的 询问。

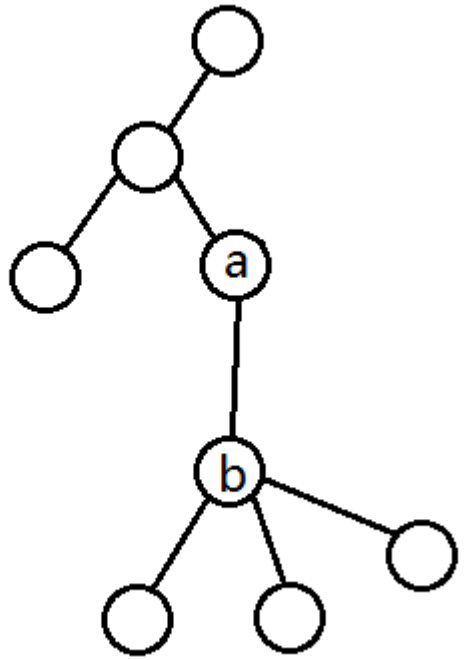
Solution

- 可以发现每次查询点 a ， b 两端构成的简单路径个数
- 即等价于 a 不经过 b 的子树大小和 b 不经过 a 的子树大小的乘积
- 因为任意左边一个点和右边一个点都有唯一的而且不同的一条路径



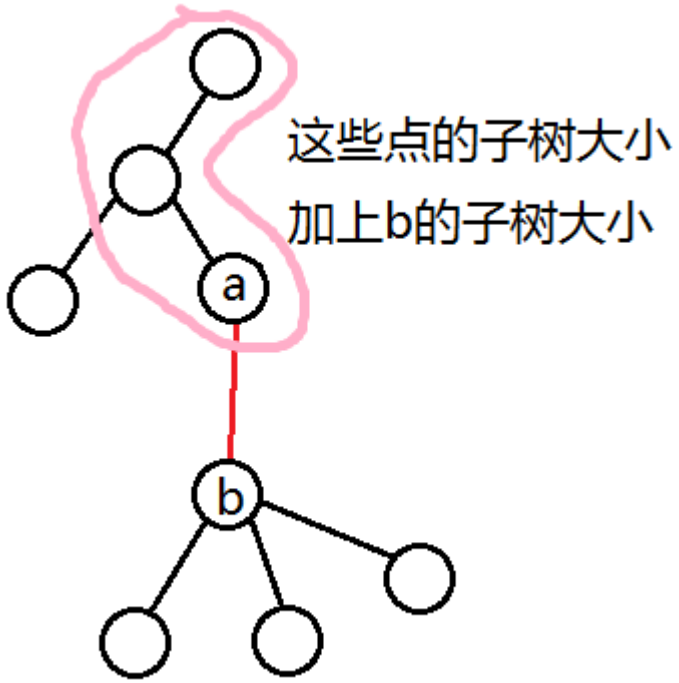
Solution

- 于是考虑离线，先把这棵树建出来，每个点维护子树大小
- 假设 a 是 b 的父亲，则 a 和 b 之间路径的答案为 (a 所在联通块大小 -b 子树大小)* b 子树大小



Solution

- 每个点维护子树大小，用并查集维护每个联通块的大小
- 则每次连接一条边的时候，等价于把一条链上的子树大小都加上一个值
- 比如连接 a 和 b 就是把：
- a 所在联通块里面深度最低的那个点到 a
- 这一条链加一个数
- 深度最低的那个点用并查集维护即可



Solution

- 问题转换为链加，单点求值
- 用树状数组轻松维护即可
- $O(\ q\log n\ )$

Luogu5354 [Ynoi2017] 由乃的 OJ

- 给你一个有 n 个点的树，每个点的包括一个位运算 opt 和一个权值 x，位运算有 &,|,^ 三种，分别用 1,2,3 表示。
- 每次询问包含三个数 x,y,z，初始选定一个数 v。然后 v 依次经过从 x 到 y 的所有节点，每经过一个点 i，v 就变成 v opti x i，所以他想问你，最后到 y 时，希望得到的值尽可能大，求最大值？给定的初始值 v 必须是在 [0,z] 之间。
- 每次修改包含三个数 x,y,z，意思是把 x 点的操作修改为 y，数值改为 z

Solution

- 如果我们知道每个位经过链上的变化之后变成了什么，那就可以贪心处理了
- 直接 HLD+ 拆位维护的话是 O(nlog^2nlogv)
- 用静态 LCT+ 拆位维护的话是 O(nlognlogv)
- 实际上不用拆位，可以压位来处理，每次处理所有的 0 位变成什么， 1 位变成什么即可
- 这样复杂度 O(nlogn)

bzoj 4771

- 一棵树，询问一个点的子树中与其距离不超过 d 的点有多少种不同的点权。
- n,m<=5e5,d 每次给出 ,5s

Solution

- 每个节点维护一个以深度为关键字的数据结构，如线段树
- 然后从下往上进行线段树合并
- 因为链分治的复杂度是 O(nlogn)，线段树合并的复杂度是插入一个元素 O(logn) 的，每次会把一段深度区间进行区间 +1

- 所以总复杂度是 O(nlog^2n) 的，使用长链剖分可能可以做到 O(nlogn)

经典问题（好像是 51nod 上面的）

- 给出一棵树，边权为 1 ，每次询问给两个 点编号的区间，求从两个区间中各选出一个点能得到的树上最远距离。
- 就是从 [l1,r1] 中选一个 a ， [l2,r2] 中选一个 b ，求 max dist(a,b)

Solution

- 直径的性质： A 集合中 a 到 b 最远， B 集合中 c 到 d 最远，这里有很多个直径的话只用选其中一个
- 则 A 和 B 的并集中直径是从这四个点里面选两个构成的
- 线段树维护区间直径端点即可
- 总合并次数 O(mlogn)，如果使用 O(nlogn) - O(1) 的 rmq
- 可以做到 O((n+m)logn)，这里不带修改，所以理论上可以做到 O((n+m)α(n))

Codechef DGCD

- 给出一棵 N 个点点权树，有 M 次操作:
  1、询问一条路径上的 GCD，即最大公约数
  2、将一段路径上的点权加上 d

Solution

- 在序列上如何维护呢?
- gcd(a , b) = gcd(a - b , b)
- 将每个位置差分:
- b[i] = a[i-1]-a[i]
- 则 a 的区间加对应了 b 的单点修改
- a 的区间 gcd 和 b 的区间 gcd （特判端点）相同

Solution

- 所以我们可以维护差分后的树，这里可以把每个位置和其父亲差分，也可以在树链剖分的 DFS 序上差分，区间加变成了单点修改，就可以维护了，注意需要特判端点
- 复杂度和如何实现有关

Wannafly 挑战赛 13F

- 维护一棵 N 个结点的无根树。 支持两种操作:
- 1. 在链 (u,v) 的每个点上放一个可爱值为 k 的 fafa。
- 2. 问所有可爱值在 [l,r] 内的 fafa 到点 p 的距离之和。
- N,Q≤ 10^5 。

Solution

- 链插入点？点插入链！
- 维护一个点集，支持加链，询问一个点到点集内所有点的距离和
- 距离和可以用之前讲的方法转化为 lca 深度和
- 所以就是链加等差数列，链和

Solution

- 每次查询是查询可爱值在 [l,r] 内的 fafa 到点 p 的距离之和，也就是点到区间的点集的距离和
- 所以这里加上一层线段树的分治，用线段树套树链剖分 + 线段树，或者树链剖分 + 树套树实现，单次 O(log^3n)
- 可以用高级动态树做到 O(log^2n)

Codeforces 757G

- 给出一棵 n 个点的树及一个 1~n 的排列 pi ，边有边权，有 q 次操作:
- 1 l r x 求 ∑ i=dis(p_i,x) , l <= i <= r
- 2 x swap(p_x,p_{x+1}) n,q<=2 * 10^5 , 强制在线

Solution

- 点到集合的 dist 和之前讲过，可以用点到根加点到根和的方法处理，这个有 1log 做法，不过用 2log 的树链剖分 + 线段树也可以
- 不带修改的话就可持久化一下线段树，每次询问差分 [l,r]=[1,r]-[1,l-1] ，然后在两个可持久化线段树的位置上查询
- 这个 swap 相邻的操作如何处理?
- 直接换成修改?
- 但是这样会多一个 log

Solution

- 注意到我们只 swap 相邻的两个位置
- 比如 swap 了 x 和 x+1 的位置，那对 1,2,…x-1 的可持久化数据结构没有影响，由于有交换律，所以对 x+2,x+3,…n 的可持久化数据结构也没有影响
- 可以把 x-1 的可持久化数据结构插入 p[x+1]，作为 x 的可持久化数据结构

- 树链剖分 O((n+m)log^2n)，可以用之后讲的方法做到 O((n+m)logn)

Luogu2056 [ZJOI2007] 捉迷藏

- 树，点权 0 ， 1
- 每次修改一个点点权，询问两个最远 1 的距离

Solution

- 对于传统树分治做法，就是在每个树分治的分治中心上开个堆维护一下：
- 每个子树一个堆维护到其的最大距离
- 再开个堆维护上面那个堆的前 2 大值
- 再开个堆维护全局的最大值
- 然后每次暴力更新即可，总复杂度 $O((n+m)\log^2 n)$

Solution

- 还有一种括号序列的做法
- 本质上是 ETT 在这个题上的特殊化，这里就不介绍了

Bzoj2566 xmastree

- 有一棵含 N 个结点的树，树上每条边 (ai,bi) 都有一个权值 wi 。树上每个结点涂有一个初始颜色 ci 。现在有很多次修改操作，第 i 次修改会将结点 xi 的颜色修改成 yi 。请在所有修改前和每次修改之后输出一个数，表示对应时刻最近的同色结点对的距离。
- 其中，距离定义为树上两点的最短路的距离。最短路按边的权值 wi 计算。

Solution

- 和之前那个题差不多，对每个颜色分别开堆维护即可，大概就是 `map < int , priority_queue < int > > a` 这样的写法
- 这样 a[x] 表示的就是颜色为 x 的堆，由于建立一个空堆也是 O(1) 的，而且每次访问到 a[x] 的时候，如果要修改里面的值，代价都是 O(logn) 的，所以这里直接把 map 当数组用是不会有额外复杂度的

Luogu6329 震波

- 树，点权
- 1. 修改一个点的点权
- 2. 查询距离一个点 <=k 的所有点的点权和

Solution

- 裸题
- 开一个数据结构，维护离分治中心距离为 1... 的所有点的点权和
- 为了防止算重，可以再开一个数据结构，维护每个点到分治中心的父亲的负贡献即可差分掉

Bzoj4372 烁烁的游戏

- 树，点权
- 1. 修改距离一个点 <=k 的所有点的点权和
- 2. 查询一个点的点权

Solution

- 和前一题有啥区别

Loj 6145

- 给出一棵树，每次询问一个点 x 到编号在 [l,r] 中的点的距离的最小值。

Solution

- 先把点分树搞出来，然后对每个分治中心按点编号顺序开一棵线段树来记录每个点到分治中心的距离最小值。
  查询的话，就在该点在点分树上到根的路径中所有的线段树上查询即可。
  为什么这样是对的呢？首先因为所有点都会被算到，其次，我们虽然可能算重，把一个点多算几次，或者把一个距离算的更长，但是不会少算，而且因为 min 是具有幂等律的信息，即合并多次和合并一次等价，所以这里不会构成影响
- 由于不带修改，所以可以使用静态的 rmq 结构
- O((n+m)logn)

Luogu5311 [Ynoi2011] 成都七中

- 音无彩名给了你一棵 n 个节点的树，每个节点有一种颜色，有 m 次查询操作
- 查询操作给定参数 l r x，需输出：
- 将树中编号在 [l,r] 内的所有节点保留， x 所在联通块中颜色种类数
- 每次查询操作独立，保证 l <= x <= r

Solution

- 每次查询的时候
- 考虑在点分树上找到深度最浅的一个点，满足这个点被保留区间点 x 所在连通块包含
- 这个节点一定能被找到
- 然后我们可以把根固定为这个点来进行统计

Solution

- 转换为
- 给定一棵有根树
- 每次查询保留区间点，根所在连通块的颜色数

Solution

- 对每个颜色分别考虑贡献
- 一个点在保留区间颜色的时候和根连通等价于这个点到根路径上 [min,max] 的区间是区间 [l,r] 的子区间
- 问题就转化为了，平面上有若干个点，询问某个点左下角的点的颜色数
- 这个可以离线 O(logn) 解决，前面点分治是 O(logn) 代价

- 总复杂度 O(nlog^2n)

floj307 不可知圆环

Description

纱绫发现，『不可知圆环』不仅可以进行传送，如果在传送中关闭，还可以将物品切断。

纱绫决定试验一下这个能力。她找到了一个 n 个节点的树，对于每条边都决定切断或不切断，这样一共有 2^{n-1} 种方案。纱绫想知道，有多少种方案使得点 1 所在连通块的大小为 k。答案对 1811939329 取模。

- 输出 k=0~n+1 的所有答案
- 大概是 1e5,5s

Solution

- 如何合并两个子图的答案?
- f[i] 表示合并上来的连通块大小为 i 的方案数
- 用 f1 和 f2 合并出 g：
- g[i] = sigma f1[i] * f2[i-j]
- 这个实际上就是
- for(i , j) g[i+j] += f1[i] * f2[j]
- 是一个卷积形式，考虑树分治 +FFT 维护

Solution

- 处理树的问题通常可以考虑分治，而点分治在这道题中会有一些不便，考虑进行链分治。
- 先对树进行重链剖分，每次处理一条重链。
- 先递归下去求出每个轻儿子的答案数组。对于一个重链上的点，合并其轻儿子可以用分治 FFT 做到 O(nlog^2n)

Solution

- 这样每个重链上的点都有一个答案数组。
- 考虑合并重链上的点。如果按深度从小到大对重链编号，则连通块在重链上的部分一定是一段前缀，可以对重链进行分治 FFT。一个前缀要么是左半的一部分，要么是左半加右半的一部分，所以分治时可以维护两个数组，一个是包含左端点的答案，一个是同时包含左右端点的答案。
- 时间复杂度为 $O(n\log^3 n)$，但常数很小。

Solution

- 树分治 + 重链分治我们会想到什么?
- 树链剖分
- 树链剖分是 2log 的，因为划分不均匀
- 可以使用更优的结构做到 1log
- 于是这题可以做到 2log

静态 lct

- 很多问题并不涉及到树形态的修改，所以用 lct 维护会常数比较大
- 可以考虑将 lct 给”静态化”
- 发现 HLD 的问题是虽然跳轻边的次数是 O(logn) 的，但是在重链上表现不佳
- 因为 HLD 是对于每条重链局部平衡的，并不能和全局的结构很好地配合起来

静态 lct

- 对于重链我们不建一个平衡的线段树，而是建一个基于 biased dictionary problem 的 bst
- 将每个节点赋予额外权值为其轻儿子子树 size 和 +1
- 对于每条重链我们找该重链的带权重心，以该节点作为 bst 的根，两边做为左右子树建 bst
- 轻边不改变
- 可以证明这样的结构我们每次跳 a 层，子树 size 一定会增大 1/b，其中 a 和 b 是两个常数，好像是 bst 上跳三层一定增大一倍，所以深度是 O(logn) 的
- 这个挺好写的，缺点是静态结构无法动态改变树的形态

Luogu5314 [Ynoi2011]ODT

- 给一棵树，支持：
- 1. 把一条链上所有点加上 k
- 2. 查询距离一个点 <=1 的所有点的点权 kth

Luogu5314 [Ynoi2011]ODT

- 对这题的评价：(5-8)/11

Solution1

- 暴力

- 总复杂度 O(nm)
- 期望得分: 20

Solution2

- 对点的度数进行根号分治
- 度数 >sqrt(n) 的点有 sqrt(n) 个，其他点度数都 <sqrt(n)
- 维护所有大点的数据结构
- 每次修改 O(sqrtn * 数据结构复杂度)

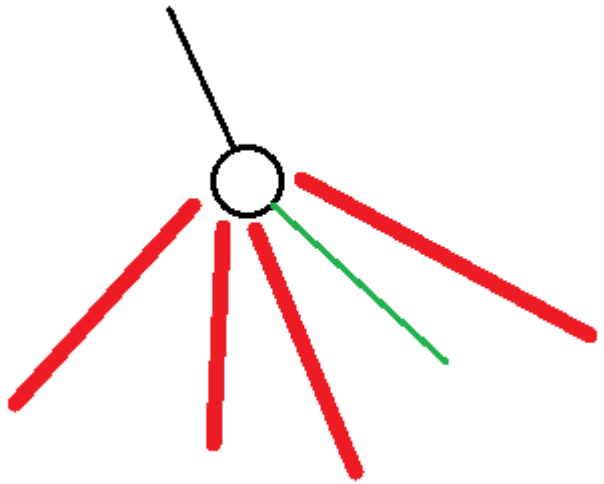
Solution2

- 查询的时候如果是大点就直接查，如果是小点就是暴力查询一圈
- 这个是 O(sqrt(n) * 数据结构复杂度)
- 对两个数据结构都用一个 O(sqrt(n)) - O(1) 平衡的分块
- 于是做到了 O(msqrt(n)) 的复杂度

- 期望得分: 20-50

Solution3

- 考虑链分治
- 一个点距离 <=1 的点里面，只有 3 个可能不是重链头
- 自己，父亲，轻儿子



Solution3

- 然后每次查询的时候暴力插入这 3 个（O(1) 个）点
- 每次修改的时候把一条链上所有重链头的父亲都修改这个重链头的值
- 于是做到了查询 O(logn)
- 修改 O(log^2n)
- 的复杂度，期望得分: 80
- 看看能不能继续优化?

Solution4

- 考虑查询一个点的时候再处理经过这个点的所有修改
- 直觉上这样可以减少很多修改

Solution4

- 这样的复杂度似乎是均摊 O((n + m)logn)?
- 证明 1：
- 这个问题可以转换为:
- 有一颗边全是黑色的树
- 每次可以把一条链染色为黑色
- 或者把一个点一圈全染色为白色，复杂度为一圈黑色的点数
- 证明: 复杂度为 O(n + m)

Fact

- 我问了一群大爷，大家都从直觉上觉得这个东西肯定是 O(n + m) 的，但是都无法给出严谨的证明。。。

God

oscar

搞一棵x层x叉树

oscar

每当有一个叶子到根的路径都是白的

oscar

就把她搞黑

oscar

每当有一个节点所有指向儿子的边都是黑的

oscar

就把她搞白

oscar

然后可以n*x次内每次都改变至少x条边的状态

oscar

并且最终状态和初始状态一样

oscar

😔😔😔

oscar

这样对不对啊

0h no

- 我们的直觉其实很有可能是错的

WTF

- 那这样的话按照复杂度证明 1，这个的复杂度是什么呢?
- 相当于求 x^x 的反函数
- 这个的复杂度是 e^W(ln(n)),W 是 ProductLog
- 然后可以证明出
- e^W(ln(n)) = Θ(logn/loglogn)

$$\left(\frac{\ln x}{\ln(\ln x)}\right)^{\frac{\ln x}{\ln(\ln x)}}$$

$$= e^{\frac{\ln x}{\ln(\ln x)}\ln\left(\frac{\ln x}{\ln(\ln x)}\right)}$$

$$= e^{\frac{\ln(x)\left(\ln(\ln x)-\ln(\ln(\ln x))\right)}{\ln(\ln x)}}$$

$$= e^{\ln(x)\left(1-\frac{\ln(\ln(\ln x))}{\ln(\ln x)}\right)}$$

$$= x^{1-\frac{\ln(\ln(\ln x))}{\ln(\ln x)}}$$

Solution4

- 如果按照这个问题来算
- 这个做法的复杂度是 Ω ((n + m)log^2n/loglogn)

- 期望得分: 100

Solution5

- 这个问题比原问题强，进一步弱化
- 有一颗 n 个点的树，边有颜色，初始全黑，支持把一条链上的边染黑，或者把一个点相邻的边染白 (代价为 相邻黑边数 *log (度数)) 求证 n 次操作代价为 O(nlogn)

Solution5

- 可以看到那个 hack 数据对这个做法的复杂度是:
- O(logn/loglogn * log(logn/loglogn)) < O(logn/loglogn * loglogn) = O(logn)
- 失效了

Preknownledge

- 多序列 kth 的做法:
  设有 m 个序列，长度为 a[1],a[2], ... ,a[m]，由 m 棵平衡树分别维护，希望查出第 k 小的元素
  要求序列中不含相同元素
  不妨设 k≤a[1]+a[2]+...+a[m]

Preknowledge

- 平衡树支持 O(1) 找出 rank 在 [71%,80%] 之间的某个元素（不关心具体 rank，只要在范围内即可），并按这个元素 O(1) 分裂成比它小 / 大的部分（不包含这个元素）

Preknowledge

- 将每棵平衡树按对应序列长度大小加权，以某个 rank 在 [71%, 80%] 的元素为 key ， O(m) 求加权 nth_element ，将 m 棵树分成 3 组，第二组只有一棵树，第三组的 key 比第二组大，第一组的 key 比第二组小，第 1,2 组的序列长度之和至少占总序列长度之和的 71% ，第 2,3 组的序列长度之和至少占总序列长度之和的 29% 。于是第 1,2 组中小于等于本组 key 的元素个数占 m 个序列长度之和的 50% 以上，且这些元素都比第二组的 key 小；第 2,3 组中大于等于本组 key 的元素个数占 m 个序列长度之和的 5.8% 以上。将第 2,3 组中平衡树替换为小于本组 key 的部分，得到子问题，而总的序列长度减少了至少 5.8% 。
  因此复杂度是 O(mlog(a[1]+a[2]+...+a[m]))

Solution5

- 对每个点，将轻边按子树 size 降序排序，分组，每组依次包含 2,4,16,256,65536, ... 条轻边，每组用平衡树维护
  查 kth 的时候使用多序列 kth 的 trick，复杂度是 logn 的
  链加的时候特判重链，可以发现轻边上改平衡树的复杂度是 log n 的

Solution5

- 第 k 组的轻边至多有 2^(2^(k-1)) 条
  轻边为第一组的情况修改是 O(1) 的，这种情况每次链加只有 O(logn) 次
  轻边不在第一组时，自顶向下经过第 k 组轻边，则子树大小减小至不超过原来的 1/(2^(2^(k-2)))，修改复杂度是 O(2^(k-1))，以 log2(子树大小) 为势能可以证明单次链加的复杂度为 O(logn)
  重链部分把链加点查转化为点加子树求和，也是 O(logn) 的
- 时间复杂度 O((n + m)logn)
- 空间复杂度 O(n + m)
- 期望得分: 80-100

Thanks for listening