动态树相关

吴作凡

安徽师范大学附属中学

2016年5月16日

前言

动态树问题是一类经典的数据结构试题，让你动态维护一些树上的信息，一些基本的操作有：

1. 加边或者删边；
2. 对树上一条路径的修改或者询问；
3. 对一个子树的修改或者询问。

在OI竞赛中，动态树问题经常出现，但这些试题基本已经形成一些套路，作为一名合格的OIer当然是要会这些套路的啦！

静态树问题

很多试题中树的形态都不会变化，只要你对路径或子树进行修改或询问，这类问题我们可以称为静态树问题。

静态树问题

很多试题中树的形态都不会变化，只要你对路径或子树进行修改或询问，这类问题我们可以称为静态树问题。

解决这类问题的方法一般是轻重链剖分。

子树操作

如果只存在子树操作，我们该怎么办呢？

子树操作

如果只存在子树操作，我们该怎么办呢？

在dfs一棵树的过程中，肯定会先遍历完一棵树的子树，再访问其他点。所以对于任意一个点，它的子树在dfs序中都是一个连续区间，那么我们求出dfs序，然后用数据结构维护就好啦！

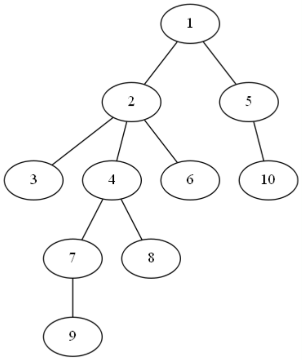
路径操作

如果存在路径操作，我们该怎么办呢？

路径操作

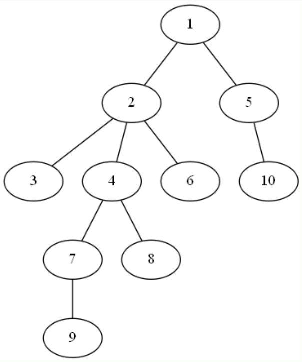
如果存在路径操作，我们该怎么办呢？
啊。。我太弱了什么也不会！先把dfs序弄出来再说吧！

路径操作



dfs序就是1,2,3,4,7,9,8,6,5,10咯！

路径操作



dfs序就是1,2,3,4,7,9,8,6,5,10咯！

仔细观察一下这个dfs序，我们发现里面存在1-2-3,4-7-9,6,5-10这些路径！也就是说dfs序是由一些路径组合起来的！

路径操作

实际上，某个点的第一个被访问的子树会和它组合成一条路径。我们把这棵树剖成了许多路径，一条路径就可以分解为若干条这些路径的子段，那么我们就需要找到一个尽量优的剖分方法，使得每条路径都可以分解为尽量少的段。

路径操作

实际上，某个点的第一个被访问的子树会和它组合成一条路径。我们把这棵树剖成了许多路径，一条路径就可以分解为若干条这些路径的子段，那么我们就需要找到一个尽量优的剖分方法，使得每条路径都可以分解为尽量少的段。

怎么分呢？随机一波？复杂度好像不怎么靠谱啊，怎么卡？

路径操作

实际上，某个点的第一个被访问的子树会和它组合成一条路径。我们把这棵树剖成了许多路径，一条路径就可以分解为若干条这些路径的子段，那么我们就需要找到一个尽量优的剖分方法，使得每条路径都可以分解为尽量少的段。

怎么分呢？随机一波？复杂度好像不怎么靠谱啊，怎么卡？

按照子树大小加权随机？讲得好，可这毫无意义！直接按照较大的子树剖就好啦！这就是轻重链剖分。

轻重链剖分

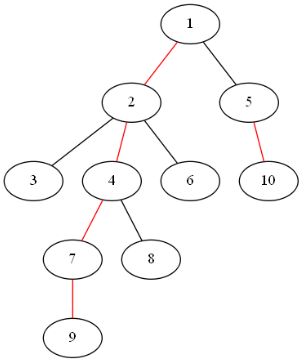
一个点的子树最大的儿子我们称为重儿子，那条边就叫做重边，其它就是轻儿子和轻边了。一条极大的只由重边组成的链我们称为重链。显然一条路径只会包含$O(\log n)$条轻边和重链的子段。

轻重链剖分

一个点的子树最大的儿子我们称为重儿子，那条边就叫做重边，其它就是轻儿子和轻边了。一条极大的只由重边组成的链我们称为重链。显然一条路径只会包含$O(\log n)$条轻边和重链的子段。

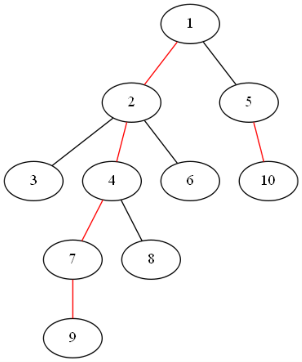
实现的时候就先dfs一遍求出重儿子，再dfs一遍求dfs序，优先访问重儿子就好啦！这样每条重链都会是连续一段，那么每条路径都会被划分成$O(\log n)$个区间。用数据结构维护dfs序就好啦！

轻重链剖分



红色的边就是重边。

轻重链剖分



红色的边就是重边。
dfs序就是1,2,4,7,9,8,3,6,5,10。

spoj QTREE

给一棵n个点的带边权的树，每次询问一条路径的最大边权或者修改一条边边权。

spoj QTREE

给一棵n个点的带边权的树，每次询问一条路径的最大边权或者修改一条边边权。

边权可以转化为子节点的点权，然后直接裸上轻重链剖分，用线段树维护一下，复杂度$O(m\log^2 n)$。

Codechef QTREE

给一棵n个点的带边权的基环树，环的大小是奇数，m次操作，将一条最短路径取反或者询问最短路径的最大子段和。$n, m \leq 10^5$

Codechef QTREE

给一棵n个点的带边权的基环树，环的大小是奇数，m次操作，将一条最短路径取反或者询问最短路径的最大子段和。$n, m \leq 10^5$

如果是树很好处理，基环树我们就断掉环上一条边，再判断一下就好了。复杂度$O(m \log^2 n)$。

spoj QTREE6

给一棵n个点的树，每个点是黑或白色，m次操作，询问每个点所在同色连通块大小或者修改某个点的颜色。$n, m \le 10^5$

spoj QTREE6

给一棵n个点的树，每个点是黑或白色，m次操作，询问每个点所在同色连通块大小或者修改某个点的颜色。$n, m \leq 10^5$

用$f[u][0/1]$记录一个点子树中黑白色连通块大小，修改只会修改一条链的权值，直接剖分维护一下就好。复杂度$O(m \log^2 n)$。

Hnoi2016 网络

给一棵n个点的树，一个交互请求存在于一条路径上，有一个优先级，m次操作，加入或者删除一个请求，或者询问删去某个点还存在的最高优先级。$n \le 10^5, m \le 2 * 10^5$

Hnoi2016 网络

给一棵n个点的树，一个交互请求存在于一条路径上，有一个优先级，m次操作，加入或者删除一个请求，或者询问删去某个点还存在的最高优先级。$n \leq 10^5, m \leq 2 * 10^5$

一条路径会被剖成$O(\log n)$个区间，那么补集也只会有$O(\log n)$个，那么问题就转化为了区间的问题。删除不好处理可以离线分治，复杂度$O(m \log^2 n)$。这个做法和CTSC day1T1很像。

本题还存在$O(m \log n)$的算法，不过和轻重链剖分无关，就不讲啦。

Sub

给一棵n个点的有点权的树，m次操作，修改一个点的点权或者询问最大连通块。$n, m \leq 10^5$

Sub

给一棵n个点的有点权的树，m次操作，修改一个点的点权或者询问最大连通块。$n, m \leq 10^5$

如果是一条链，直接线段树维护最大子段和就好啦！

Sub

给一棵n个点的有点权的树，m次操作，修改一个点的点权或者询问最大连通块。$n, m \leq 10^5$

如果是一条链，直接线段树维护最大子段和就好啦！

如果是一条树链，每个点的权值就是只考虑轻儿子的最大连通块的值，对于每条链求最大子段和就好啦！实现的时候可以在相邻两个重链之间插入负无穷作为分隔符。修改点权只会对$O(\log n)$的权值产生影响。复杂度$O(m \log^2 n)$。

UOJ#191 Unknown

给你一个初始为空的序列S，m次操作，在末尾加入或删除一个向量，询问$S[L...R]$中的向量和x的最大叉积。$m \leq 5 * 10^5$

UOJ#191 Unknown

给你一个初始为空的序列S，m次操作，在末尾加入或删除一个向量，询问$S[L...R]$中的向量和x的最大叉积。$m \leq 5 * 10^5$

在末尾插入或者删除我们可以建出操作树，那么就变成询问树上一条自下而上的链中的最大叉积。而我们知道最大叉积一定是在凸包上。

UOJ#191 Unknown

给你一个初始为空的序列S，m次操作，在末尾加入或删除一个向量，询问$S[L...R]$中的向量和x的最大叉积。$m \leq 5 * 10^5$

在末尾插入或者删除我们可以建出操作树，那么就变成询问树上一条自下而上的链中的最大叉积。而我们知道最大叉积一定是在凸包上。

观察剖分成的区间的性质，我们发现除了最上面的一个区间，其它都是某条链的一个前缀！那么对于$O(m)$个最上面区间我们可以通过一次分治求解，其余部分可以用平衡树动态维护凸包。

时间复杂度$O(m \log^2 m)$，空间复杂度$O(m \log m)$，我们能不能将空间复杂度降为$O(m)$呢？

UOJ#191 Unknown

给你一个初始为空的序列S，m次操作，在末尾加入或删除一个向量，询问$S[L...R]$中的向量和x的最大叉积。$m \leq 5 * 10^5$

在末尾插入或者删除我们可以建出操作树，那么就变成询问树上一条自下而上的链中的最大叉积。而我们知道最大叉积一定是在凸包上。

观察剖分成的区间的性质，我们发现除了最上面的一个区间，其它都是某条链的一个前缀！那么对于$O(m)$个最上面区间我们可以通过一次分治求解，其余部分可以用平衡树动态维护凸包。

时间复杂度$O(m\log^2 m)$，空间复杂度$O(m\log m)$，我们能不能将空间复杂度降为$O(m)$呢？

我们可以将$O(m\log m)$次前缀询问分成$O(\log m)$组，这样就将空间降为$O(m)$啦！然而并没有什么卵用。

动一动

轻重链剖分能不能动起来？给一棵树，n次操作，每次会加入一个叶子，要你动态维护这个剖分的结构，每次输出重儿子的编号和。

动一动

轻重链剖分能不能动起来？给一棵树，n次操作，每次会加入一个叶子，要你动态维护这个剖分的结构，每次输出重儿子的编号和。

给某个点加个叶子，子树大小只会增加，那么只可能它的轻边变成重边，修改一下就好了。

动态树问题

在更加难的题中，存在一些加边和删边操作，并要求你对路径或子树进行修改或询问，这类问题我们可以称为动态树问题。

动态树问题

在更加难的题中，存在一些加边和删边操作，并要求你对路径或子树进行修改或询问，这类问题我们可以称为动态树问题。

为了解决这类问题我们需要一些动态树算法，常用的算法有Link-Cut Tree，Euler Tour Tree和Self-Adjusting Top Tree。

Link-Cut Tree

LCT的主要思想就是动态维护剖分，所以对于路径操作它是一个非常有效的算法。

一些定义

因为树的结构变动相当大，我们不能再按照子节点大小来划分轻边重链，LCT采取了一种非常高明的策略，在了解这个策略之前当然要做一点定义。

一些定义

因为树的结构变动相当大，我们不能再按照子节点大小来划分轻边重链，LCT采取了一种非常高明的策略，在了解这个策略之前当然要做一点定义。

Preferred Child:偏爱子节点，一个点x的偏爱子节点y满足，最后一次访问x子树中的点是在y的子树中。显然一个点的偏爱子节点最多只有一个。

Preferred Edge:偏爱边，连接点和偏爱子节点的边就是偏爱边。

Preferred Path:偏爱路径，一条极长的仅由偏爱边组成的路径称为偏爱路径。

Access

LCT的最核心操作就是访问一个节点（Access）。根据之前的定义，我们知道如果我们访问一个节点，我们会将它到根的路径变成一条偏爱路径，修改一些点的偏爱子节点。

Access

LCT的最核心操作就是访问一个节点（Access）。根据之前的定义，我们知道如果我们访问一个节点，我们会将它到根的路径变成一条偏爱路径，修改一些点的偏爱子节点。

为了完成这个操作，我们要用数据结构来维护偏爱路径。轻重链剖分我们一般用线段树，而这里偏爱路径会改变，那就只能用Splay啦。

Access

LCT的最核心操作就是访问一个节点（Access）。根据之前的定义，我们知道如果我们访问一个节点，我们会将它到根的路径变成一条偏爱路径，修改一些点的偏爱子节点。

为了完成这个操作，我们要用数据结构来维护偏爱路径。轻重链剖分我们一般用线段树，而这里偏爱路径会改变，那就只能用Splay啦。

我们对每条偏爱路径都用一个Splay来维护，而顺序当然就是深度了。我们在Splay的根处记录上这条偏爱路径的父亲，这样就很容易进行Access操作了。而Access以后一个点到根的路径会变成一棵Splay，用一些基本的技巧就可以进行路径操作了。

Makeroot

很多动态树问题中都需要进行换根操作，而根据LCT的定义我们很容易完成换根操作，只需要Access以后将这个Splay完全翻转就好，利用标记来实现。

Link

如果加边(x,y)，我们就先Makeroot(y)，然后将这条偏爱路径的父亲记作x。

Cut

如果删边(x,y)，我们就先Makeroot(x)，在Access(y)并Splay(y)，然后将y和其左子树切断就好。

时间复杂度

可以看出之前所有操作中都需要Access，而除了Access以外都是$O(1)$次Splay的操作，我们知道Splay的复杂度是均摊$O(\log n)$的，于是只要分析Access的复杂度就好了。

时间复杂度

可以看出之前所有操作中都需要Access，而除了Access以外都是$O(1)$次Splay的操作，我们知道Splay的复杂度是均摊$O(\log n)$的，于是只要分析Access的复杂度就好了。

这里我们可以分两部分证明Access的复杂度，首先证明偏爱边的切换是均摊$O(\log n)$的，再证明Splay操作总和是均摊$O(\log n)$的。

时间复杂度

可以看出之前所有操作中都需要Access，而除了Access以外都是$O(1)$次Splay的操作，我们知道Splay的复杂度是均摊$O(\log n)$的，于是只要分析Access的复杂度就好了。

这里我们可以分两部分证明Access的复杂度，首先证明偏爱边的切换是均摊$O(\log n)$的，再证明Splay操作总和是均摊$O(\log n)$的。

我们将这棵树轻重链剖分，一次Access只会将$O(\log n)$条轻边切换成偏爱边，而重边则非常多。将重边切换为偏爱边和将重边切换为非偏爱边的级别相同，而将重边切换为非偏爱边肯定会有轻边切换为偏爱边，那么切换次数就均摊$O(\log n)$了。

时间复杂度

可以看出之前所有操作中都需要Access，而除了Access以外都是$O(1)$次Splay的操作，我们知道Splay的复杂度是均摊$O(\log n)$的，于是只要分析Access的复杂度就好了。

这里我们可以分两部分证明Access的复杂度，首先证明偏爱边的切换是均摊$O(\log n)$的，再证明Splay操作总和是均摊$O(\log n)$的。

我们将这棵树轻重链剖分，一次Access只会将$O(\log n)$条轻边切换成偏爱边，而重边则非常多。将重边切换为偏爱边和将重边切换为非偏爱边的级别相同，而将重边切换为非偏爱边肯定会有轻边切换为偏爱边，那么切换次数就均摊$O(\log n)$了。

而Splay一个点x的复杂度其实是$O(\log n - \log sz_x)$的，那么加一加抵一抵就会发现是均摊$O(\log n)$啦。

时间复杂度

可以看出之前所有操作中都需要Access，而除了Access以外都是$O(1)$次Splay的操作，我们知道Splay的复杂度是均摊$O(\log n)$的，于是只要分析Access的复杂度就好了。

这里我们可以分两部分证明Access的复杂度，首先证明偏爱边的切换是均摊$O(\log n)$的，再证明Splay操作总和是均摊$O(\log n)$的。

我们将这棵树轻重链剖分，一次Access只会将$O(\log n)$条轻边切换成偏爱边，而重边则非常多。将重边切换为偏爱边和将重边切换为非偏爱边的级别相同，而将重边切换为非偏爱边肯定会有轻边切换为偏爱边，那么切换次数就均摊$O(\log n)$了。

而Splay一个点x的复杂度其实是$O(\log n - \log sz_x)$的，那么加一加抵一抵就会发现是均摊$O(\log n)$啦。

所以之前说LCT的策略非常高明，它的复杂度甚至比轻重链剖分还要优秀（如果你不去把线段树换成Splay或者全局平衡二叉树的话）!

bzoj2049 洞穴勘测

维护森林，存在加边和删边操作，询问两点连通性。

bzoj2049 洞穴勘测

维护森林，存在加边和删边操作，询问两点连通性。
直接裸上LCT就好啦！

bzoj2759 一个动态树好题

有n个未知数和方程，每个都是$x_i = k_i x_{p_i} + b_i \mod 10007$，$m$次操作，询问$x_a$的解或者修改一个方程。$n, m \le 10^5$

bzoj2759 一个动态树好题

有n个未知数和方程，每个都是$x_i = k_i x_{p_i} + b_i \mod 10007$，$m$次操作，询问$x_a$的解或者修改一个方程。$n, m \leq 10^5$

这显然是个基环森林，删去一条边变成树，用LCT维护，然后在根的位置记录一下这条边就好啦。

Dynamic Connectivity

给你n个点的无向图，加边删边询问两点是否连通，可以离线。

Dynamic Connectivity

给你n个点的无向图，加边删边询问两点是否连通，可以离线。
每条边存在时间是一个区间，分治+并查集，复杂度$O(n\log^2 n)$。

Dynamic Connectivity

给你n个点的无向图，加边删边询问两点是否连通，可以离线。
每条边存在时间是一个区间，分治+并查集，复杂度$O(n\log^2 n)$。
直接用LCT维护删除时间的最大生成树，边权如何转化为点权？

Dynamic Connectivity

给你n个点的无向图，加边删边询问两点是否连通，可以离线。
每条边存在时间是一个区间，分治+并查集，复杂度$O(n\log^2 n)$。
直接用LCT维护删除时间的最大生成树，边权如何转化为点权？
加入虚点就好了。复杂度$O(n\log n)$。

bzoj4025 二分图

给你n个点的无向图，加边删边询问图是否是二分图。

bzoj4025 二分图

给你n个点的无向图，加边删边询问图是否是二分图。
每条边存在时间是一个区间，分治+并查集，复杂度$O(n\log^2 n)$。

bzoj4025 二分图

给你n个点的无向图，加边删边询问图是否是二分图。

每条边存在时间是一个区间，分治+并查集，复杂度$O(n\log^2 n)$。

直接用LCT维护删除时间的最大生成树，如果一条边会产生奇环就加到一个集合中，集合为空就是二分图。复杂度$O(n\log n)$。

Codechef PUSHFLOW

n个点的图，每个点最多属于一个简单环，边有边权，m次操作，询问两个点的最大流或者修改一条边的边权。$n, m \leq 10^5$

Codechef PUSHFLOW

n个点的图，每个点最多属于一个简单环，边有边权，m次操作，询问两个点的最大流或者修改一条边的边权。$n, m \leq 10^5$

最大流就是最小割，只可能割环上两条边或者一条环间的边，这两条边一定会包含最小的那一条，割掉它并把它的边权加到其它边上。复杂度$O(m \log n)$。

51nod1576 Tree and Permutation

一棵m个点的树，你需要确定一个1到n的排列P使得$\sum_{i=1}^{n} \frac{dis(i,P_i)}{2}$最大，求出当$n = 1, 2...m$的答案（保证一定是个树，也就是依次加入叶子）。$m \leq 10^5$

51nod1576 Tree and Permutation

一棵m个点的树，你需要确定一个1到n的排列P使得$\sum_{i=1}^{n}\frac{dis(i,P_i)}{2}$最大，求出当$n=1,2...m$的答案（保证一定是个树，也就是依次加入叶子）。$m \le 10^5$

如果选个点当根，直观上我们想让i和P_i尽量远，也就是Lca尽量靠近根，而选择重心当根显然可以保证Lca全部在根上。那么就是要你动态维护所有点到重心的距离和。

51nod1576 Tree and Permutation

一棵m个点的树，你需要确定一个1到n的排列P使得$\sum_{i=1}^{n} \frac{dis(i,P_i)}{2}$最大，求出当$n=1,2...m$的答案（保证一定是个树，也就是依次加入叶子）。$m \le 10^5$

如果选个点当根，直观上我们想让i和P_i尽量远，也就是Lca尽量靠近根，而选择重心当根显然可以保证Lca全部在根上。那么就是要你动态维护所有点到重心的距离和。

显然加入一个叶子，重心只会向那个叶子的方向移动0或1的距离，用LCT维护一下就好了，复杂度$O(m \log m)$

51nod1576 Tree and Permutation

一棵m个点的树，你需要确定一个1到n的排列P使得$\sum_{i=1}^{n} \frac{dis(i,P_i)}{2}$最大，求出当$n=1,2...m$的答案（保证一定是个树，也就是依次加入叶子）。$m \le 10^5$

如果选个点当根，直观上我们想让i和P_i尽量远，也就是Lca尽量靠近根，而选择重心当根显然可以保证Lca全部在根上。那么就是要你动态维护所有点到重心的距离和。

显然加入一个叶子，重心只会向那个叶子的方向移动0或1的距离，用LCT维护一下就好了，复杂度$O(m \log m)$

注意到可以离线，那么直接用dfs序维护子树大小会更加容易。

51nod1576 Tree and Permutation

一棵m个点的树，你需要确定一个1到n的排列P使得$\sum_{i=1}^{n}\frac{dis(i,P_i)}{2}$最大，求出当$n=1,2...m$的答案（保证一定是个树，也就是依次加入叶子）。$m \le 10^5$

如果选个点当根，直观上我们想让i和P_i尽量远，也就是Lca尽量靠近根，而选择重心当根显然可以保证Lca全部在根上。那么就是要你动态维护所有点到重心的距离和。

显然加入一个叶子，重心只会向那个叶子的方向移动0或1的距离，用LCT维护一下就好了，复杂度$O(m\log m)$

注意到可以离线，那么直接用dfs序维护子树大小会更加容易。

如果点带权求动态重心该如何处理？显然还是只会向叶子移动一些距离，在Splay上二分一下就好了。

简单的子树维护

一棵n个点的树，存在换父亲和修改点权，问某个点的子树最大点权，能用LCT做吗？

简单的子树维护

一棵n个点的树，存在换父亲和修改点权，问某个点的子树最大点权，能用LCT做吗？

我们可以对每个节点维护一个堆，把每个轻儿子（非偏爱子节点）的子树最大点权扔进去，重儿子部分就用Splay维护一下就好了。切换偏爱边的次数时$O(n\log n)$，那么复杂度就是$O(n\log^2 n)$。

Winedag's prevention

一棵n个点的树，q次操作，增加一条路径的点权，将一条路径翻转，询问路径和/最大值/最小值。$n, q \leq 10^5$

Winedag's prevention

一棵n个点的树，q次操作，增加一条路径的点权，将一条路径翻转，询问路径和/最大值/最小值。$n, q \leq 10^5$

没有翻转操作这就是一个基础的LCT题，问题就是翻转怎么搞呢？

Winedag's prevention

一棵n个点的树，q次操作，增加一条路径的点权，将一条路径翻转，询问路径和/最大值/最小值。$n, q \leq 10^5$

没有翻转操作这就是一个基础的LCT题，问题就是翻转怎么搞呢？

对于一条偏爱路径，我们使用一个Splay来维护形状，再用另一个Splay来维护值，用次序来一一对应，那么修改就可以在值树上很好地完成而不会影响到树的形态。看上去复杂度是$O(q \log^2 n)$，实际上两种Splay操作是等价的，不会影响到均摊分析，依然是$O(q \log n)$。

Codechef MONOPLOY

一棵n个点的树，一开始每个节点颜色不同，m次操作，将一个点到根的路径染成同一种新颜色，询问子树中到根路径中不同颜色的平均值。$n, m \leq 10^5$

Codechef MONOPLOY

一棵n个点的树，一开始每个节点颜色不同，m次操作，将一个点到根的路径染成同一种新颜色，询问子树中到根路径中不同颜色的平均值。$n, m \leq 10^5$

可以发现这种颜色就相当于LCT的Access操作，直接使用LCT维护，在修改偏爱子节点的时候用树状数组维护dfs序就好啦。复杂度$O(n \log^2 n)$。

ZJOI2016 D2T1 大♂森林

n棵树，一开始每棵树都只有一个0号节点，m个操作，在$[l, r]$的树的生长节点下长出一个节点，将$[l, r]$的树的生长节点改成u（如果没有这个点则不影响），询问某棵树上两个点的距离。$n \leq 10^5, m \leq 2 * 10^5$

ZJOI2016 D2T1 大♂森林

n棵树，一开始每棵树都只有一个0号节点，m个操作，在$[l,r]$的树的生长节点下长出一个节点，将$[l,r]$的树的生长节点改成u（如果没有这个点则不影响），询问某棵树上两个点的距离。$n \le 10^5, m \le 2*10^5$

我们按照下标遍历每一棵树，处理修改。第一个操作只会加或删叶子，第二个操作会把一个点的一些儿子换到另一个点上，用LCT维护一下，可以加一些虚点来处理。复杂度$O(m \log m)$。

ZJOI2016 D2T1 大♂森林

n棵树，一开始每棵树都只有一个0号节点，m个操作，在$[l, r]$的树的生长节点下长出一个节点，将$[l, r]$的树的生长节点改成u（如果没有这个点则不影响），询问某棵树上两个点的距离。$n \le 10^5, m \le 2 * 10^5$

我们按照下标遍历每一棵树，处理修改。第一个操作只会加或删叶子，第二个操作会把一个点的一些儿子换到另一个点上，用LCT维护一下，可以加一些虚点来处理。复杂度$O(m \log m)$。

当然我们还有更容易的方法——用ETT！

子树操作

路径操作我们可以轻易地使用LCT来完成，那么对于子树操作我们该用什么算法呢？

子树操作

路径操作我们可以轻易地使用LCT来完成，那么对于子树操作我们该用什么算法呢？

路径操作在静态树上我们是用轻重链剖分来解决的，将其扩展到动态树上我们就得到了LCT；在静态树上子树操作我们是通过dfs序列来解决的，那么我们能否动态维护dfs序吗？

括号序列

如果根不变动的话，加入删除点只是在dfs序中插入删除点，同样子树也就是插入或删除区间，我们用Splay或者Treap维护dfs序就好。

括号序列

如果根不变动的话，加入删除点只是在dfs序中插入删除点，同样子树也就是插入或删除区间，我们用Splay或者Treap维护dfs序就好。

为了方便找到子树位置，我们可以将一个点拆成两个构成括号序列，中间就是子树啦！

Codechef ANUDTQ

一棵n个点的树，m次操作，子树点权加一个值，加点，删子树或者询问子树权值和。$n, m \leq 10^5$

Codechef ANUDTQ

一棵n个点的树，m次操作，子树点权加一个值，加点，删子树或者询问子树权值和。$n, m \leq 10^5$

直接使用Splay维护dfs序就好啦！

Tree

将原来的题目可持久化呢?

Tree

将原来的题目可持久化呢？
用可持久化Treap维护就好啦！

Tree

将原来的题目可持久化呢？

用可持久化Treap维护就好啦！

等等。。现在我们怎么找到一棵子树的起始末尾节点？我们没法记录父亲啊！

Tree

将原来的题目可持久化呢?

用可持久化Treap维护就好啦!

等等。。现在我们怎么找到一棵子树的起始末尾节点?我们没法记录父亲啊!

我们可以使用重量平衡树来进行动态标号,再用一个可持久化数组记录标号就好啦!复杂度$O(m \log^2 n)$。

CF414E Mashmokh's Designed Problem

一棵n个点的树，每个点的儿子有顺序。m次操作，询问两个点的距离，将一个子树接到它的k层祖先上，成为它的最后一个儿子，询问从一个点开始dfs，第k层的最后一个点是什么。$n, m \leq 10^5$

CF414E Mashmokh's Designed Problem

一棵n个点的树，每个点的儿子有顺序。m次操作，询问两个点的距离，将一个子树接到它的k层祖先上，成为它的最后一个儿子，询问从一个点开始dfs，第k层的最后一个点是什么。$n, m \le 10^5$

dfs序是个非常妙的东西，任意两个相邻的点深度只会差1，那么我们可以记录区间深度最大值和最小值，就可以轻松找到深度为k层的最后一个点以及某个点的k层祖先。

CF414E Mashmokh's Designed Problem

一棵n个点的树，每个点的儿子有顺序。m次操作，询问两个点的距离，将一个子树接到它的k层祖先上，成为它的最后一个儿子，询问从一个点开始dfs，第k层的最后一个点是什么。$n, m \leq 10^5$

dfs序是个非常妙的东西，任意两个相邻的点深度只会差1，那么我们可以记录区间深度最大值和最小值，就可以轻松找到深度为k层的最后一个点以及某个点的k层祖先。

而LCA也非常好找到，dfs序中两个点之间深度最小点的父亲就是它们的LCA。复杂度$O(m \log^2 n)$。

括号序列的弊端

括号序列没法换根！那就不能愉快地LinkCut了！怎么办？

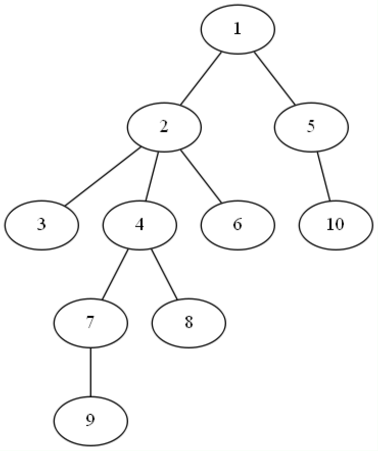
括号序列的弊端

括号序列没法换根！那就不能愉快地LinkCut了！怎么办？
我们可以使用欧拉序列。

欧拉序列

我们将树的一条边看成两条有向边，对其进行欧拉环游，得到的访问点的序列就是欧拉序列。

欧拉序列



欧拉序列就是1,2,3,2,4,7,9,7,4,8,4,2,6,2,1,5,10,5,1咯！

欧拉序列

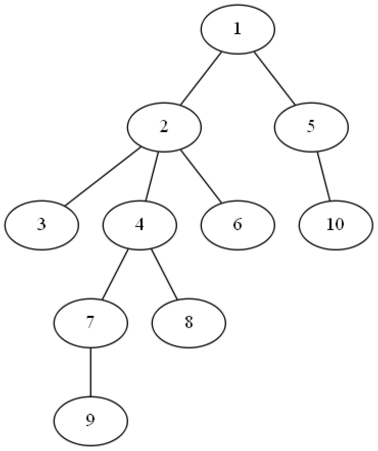
欧拉序列也有非常好的性质，比如两个点间的深度最小的节点就是LCA，一个点的第一次和最后一次出现之间就是它的子树。

欧拉序列

欧拉序列也有非常好的性质，比如两个点间的深度最小的节点就是LCA，一个点的第一次和最后一次出现之间就是它的子树。

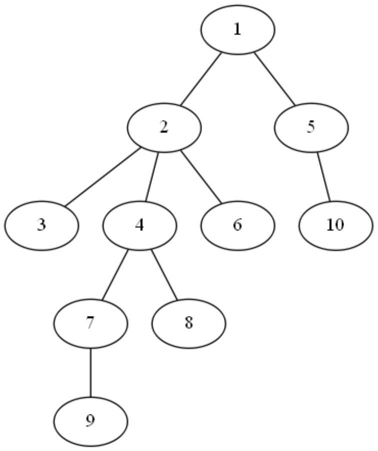
注意到欧拉序列就是一个环，所以可以完成换根操作。

换根



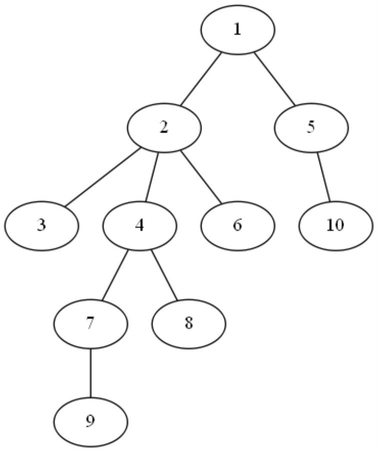
我们试着将根从1换到4。

换根



1 2 3 2 4 7 9 7 4 8 4 2 6 2 1 5 10 5 1

换根

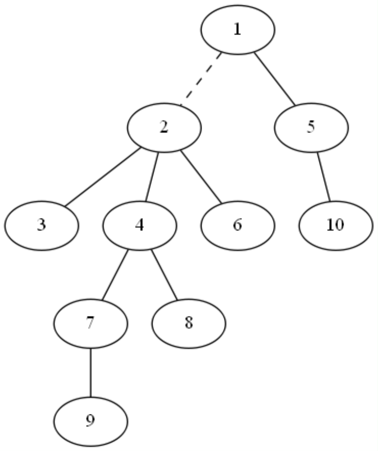


4 7 9 7 4 8 4 2 6 2 1 5 10 5 1 2 3 2 4

Link和Cut

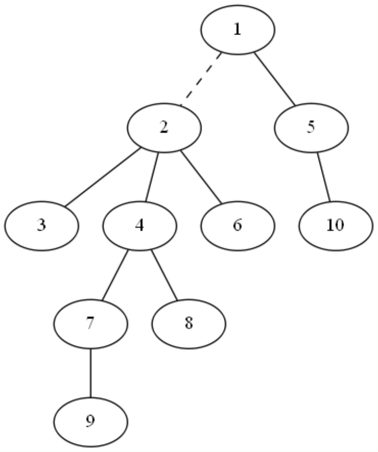
现在我们可以把一个点换为根了，那么加边删边就非常好完成了！（下面仅以Link为例）

Link



1 5 10 5 1+2 3 2 4 7 9 7 4 8 4 2 6 2

Link



1 5 10 5 1 2 3 2 4 7 9 7 4 8 4 2 6 2 1

Euler Tour Tree

于是我们可以非常容易地用Splay来维护欧拉序列，这就是ETT啦！

Euler Tour Tree

于是我们可以非常容易地用Splay来维护欧拉序列，这就是ETT啦！

可以看出欧拉序列的功能非常强大，所以我们可以试着去实现一下。。有没有发现有什么不对？

Euler Tour Tree

于是我们可以非常容易地用Splay来维护欧拉序列，这就是ETT啦！

可以看出欧拉序列的功能非常强大，所以我们可以试着去实现一下。。有没有发现有什么不对？

当存在换根的时候我们根本没法维护每个点在欧拉序列中的第一个点和最后一个点！我非常怀疑这个东西只是一个纸面上的数据结构。。感觉有点惨啊。。所以就来理性愉悦一下吧！

Dynamic Connectivity

给你n个点的无向图，加边删边询问两点是否连通，强制在线。

Dynamic Connectivity

给你n个点的无向图，加边删边询问两点是否连通，强制在线。

给每条边都打上一个等级，等级是0到$\log n$的整数，等级只会减少不会增加。

令G_i是只保留i级以下边的生成子图，F_i是G_i以等级为边权的最小生成森林，显然$F_i \subseteq F_{i+1}$。需要保证第i层的所有连通块大小不能超过2^i。

Dynamic Connectivity：Query

询问两个点是否连通，就只要在$F_{\log n}$里查询一下就好了。复杂度$O(\log n)$。

Dynamic Connectivity：Insert

加入一条边，就把它的等级赋值为$\log n$，插入$F_{\log n}$就好了。复杂度$O(\log n)$。

Dynamic Connectivity：Delete

删除一条边(u, v)，如果它不在$F_{\log n}$上，那么删去这条边不会产生任何影响，直接就结束了。

否则我们需要寻找一条边替换。

Dynamic Connectivity：Delete

删除一条边(u, v)，如果它不在$F_{\log n}$上，那么删去这条边不会产生任何影响，直接就结束了。

否则我们需要寻找一条边替换。

我们从$Level_{(u,v)}$开始，依次枚举每层寻找替换边。

假设T_u是包含u的树，T_v是包含v的树，我们令$Size_{T_u}] \le Size_{T_v}$。枚举$T_u$连出的每一条边，如果连到$T_v$则已经找到替换边并加入到$F$中，否则将这条边等级减一。

Dynamic Connectivity

这里的动态树我们选用ETT进行维护，一条边会掉$\log n$次，复杂度就是$O(\log^2 n)$。

sone1

n个点的树，m次操作，换根，链或子树加/赋值/询问min/max/sum，换父亲。$n, m \leq 10^5$。

子树操作和路径操作

如果子树操作和路径操作同时存在，我们应该怎么办呢？

子树操作和路径操作

如果子树操作和路径操作同时存在，我们应该怎么办呢？
想想轻重链剖分的时候我们是如何处理的？

子树操作和路径操作

如果子树操作和路径操作同时存在，我们应该怎么办呢？

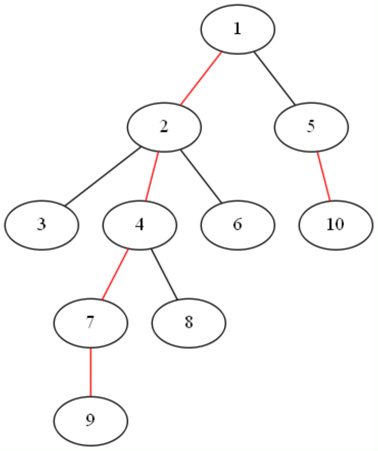
想想轻重链剖分的时候我们是如何处理的？

树剖的dfs序中子树和重链都是连在一起的，而LCT中重链是连在一起的，ETT（括号序列）中子树是连在一起的。试试将LCT和ETT结合起来？

子树操作和路径操作

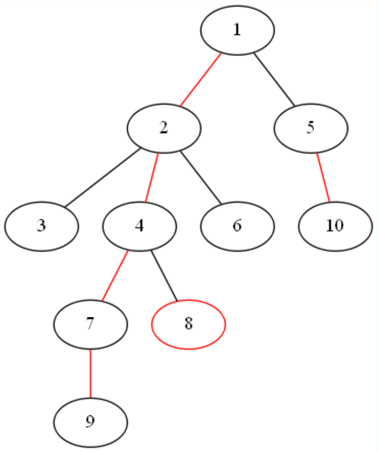
如果子树操作和路径操作同时存在，我们应该怎么办呢？
想想轻重链剖分的时候我们是如何处理的？
树剖的dfs序中子树和重链都是连在一起的，而LCT中重链是连在一起的，ETT（括号序列）中子树是连在一起的。试试将LCT和ETT结合起来？
在Access的时候，将新的偏爱子节点在ETT上移动到父亲旁边就好啦！

子树操作和路径操作



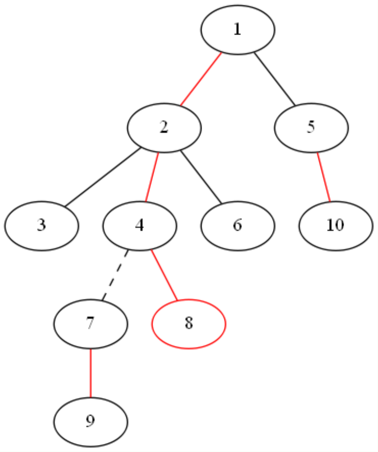
括号序列：1 2 4 7 9 9 7 8 8 4 3 3 6 6 2 5 10 10 5 1

子树操作和路径操作



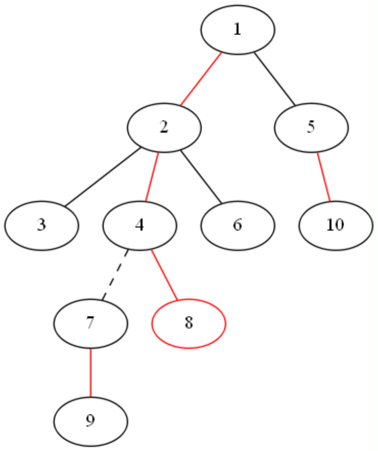
Access(8)

子树操作和路径操作



1 2 4 7 9 9 7 8 8 4 3 3 6 6 2 5 10 10 5 1

子树操作和路径操作



1 2 4 8 8 7 9 9 7 4 3 3 6 6 2 5 10 10 5 1

换根

现在我们就可以很好地Access了！但是换根需要Reverse！怎么办呢？

换根

现在我们就可以很好地Access了！但是换根需要Reverse！怎么办呢？

我们把$[1, p]$，$[p+1, 2n]$都翻转一下。但是这一条链的子树全部翻了，怎么办？

换根

现在我们就可以很好地Access了！但是换根需要Reverse！怎么办呢？

我们把$[1, p]$，$[p+1, 2n]$都翻转一下。但是这一条链的子树全部翻了，怎么办？

如果暴力遍历翻一遍显然不行，我们可以在LCT上打上标记，Access的时候看到父亲有这个标记就全部翻一发就好啦。

复杂度

这样我们就轻松解决了子树和路径的操作！复杂度$O(m\log^2 n)$。

子树操作和路径操作

如果子树操作和路径操作同时存在，有没有什么更有效的算法？

子树操作和路径操作

如果子树操作和路径操作同时存在，有没有什么更有效的算法？

我们一般使用Toptree来解决这类问题。（实际上下面讲的玩意并不是Toptree，只是非常像而已）

子树操作和路径操作

之前我们的LCT也能维护一些简单的子树信息，也就是记录虚边的信息，但是这下我们怎么下传标记呢？

子树操作和路径操作

之前我们的LCT也能维护一些简单的子树信息，也就是记录虚边的信息，但是这下我们怎么下传标记呢？

我们可以加一些节点，把虚边也建成一棵树，用Splay维护一下就好啦！

复杂度

显然这玩意和LCT完全一样，切换偏爱节点是$O(\log n)$的，考虑用Splay维护虚边的复杂度就是$O(\log^2 n)$，但是业界毒瘤Tarjan证明了这玩意是$O(\log n)$的，虽然常数是96。

Thank you!