树链剖分

算法概述

前置知识：dfs序，求lca，线段树。

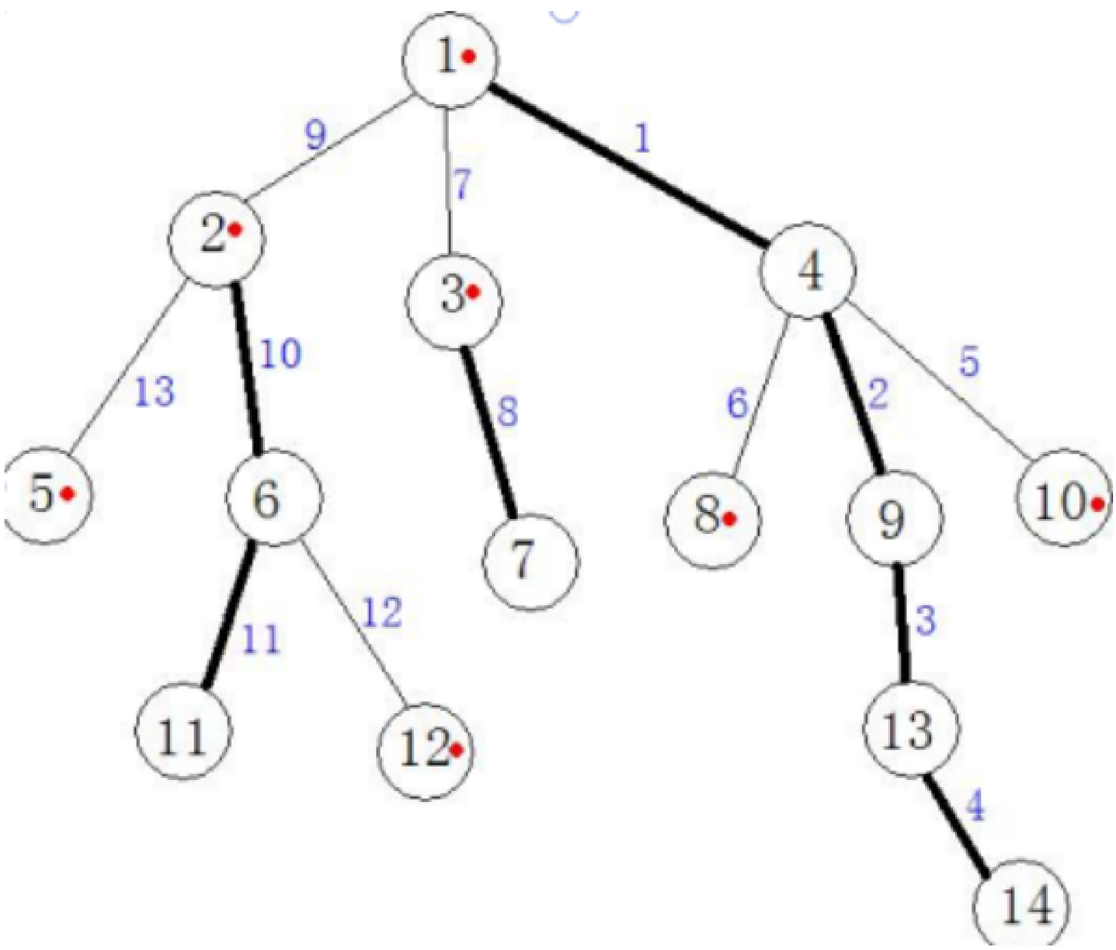
利用DFS序列和子树size将树划分成几条连，将树上问题转化为序列上线性问题。

几个概念：

- 重子节点（重儿子）：对于一个非叶子节点，重子节点是其子节点中子树最大的子结点。如果有多个子树最大的子结点，取其一。如果没有子节点，就无重子节点。
- 轻子节点（轻儿子）：除了重儿子其他的子节点
- 重边：节点到它的重儿子的边。
- 轻边：节点到它的轻儿子的边。

具体地，将一棵树划分后如下图：

其中粗线边为重边，细线边为轻边。



性质：对于任意一个点到根节点的路径上最多经过 $logn$ 条重链和轻边

换句话说树上的任意一条路径都可以表示成 $O(log(n))$ 条重链和轻边组成。

证明：任意一个点向根节点走，每走过一条轻边，则说明是由轻儿子跳到它的父亲，由于重儿子size大于轻儿子，子树大小至少翻倍，因此最多走 $logn$ 条轻边。走一次轻边即是从一条重链链顶跳到另一条重链上。

此时我们只要每条重链维护一棵线段树，单点修改时在对应线段树上修改，链查询时将要查的链拆分成 $logn$ 条重链，即可快速维护信息。

- ST表+DFS序快速求lca

类似地，当dfs进行重链剖分时每走到一个节点就先向重儿子递归，并以此为dfs序建出序列。

如上图得到的dfs序是：1 4 9 13 14 8 10 2 6 11 5 12

一条重链上的点在dfs序上是一段连续的子区间，且一个节点及其子树内所有点在dfs序上也是一段连续的子区间。

因此不需要对每条链都开一棵线段树，以dfs序建树，一条重链对应的是一个区间上的线段树。

并且由此可方便地维护链上信息和子树信息。

具体实现：

第一次dfs：划分轻重边，可以一并处理子树大小/节点深度等问题

第二次dfs：按先走重儿子再走轻儿子的顺序遍历并得到对应dfs序，一并建出线段树

　为什么是两次dfs？第一次dfs先走到的点不一定是重儿子，dfs序会改变

每个节点上要存所属重链的链顶是哪个节点，方便快速找到dfs序上对应的序列。

核心操作代码实现：

```
dep：深度   size：子树大小    fa：父亲节点      son：重儿子
用的链式前向星，line[u]是一个vector，这样写可以快速遍历其中的值
void dfs1(int u){
    dep[u]=dep[fa[u]]+1;
    size[u]=1;
    for(int to:line[u]){
        if(to==fa[u])continue;
        fa[to]=u;
        dfs1(to);
        if(size[to]>size[son[u]])son[u]=to;
        size[u]+=size[to];
    }
    return ;
}
dfn：dfs序     p：某个点对应在dfs序中的下标      top：所属重链的链顶节点
void dfs2(int u){
    dfn[u]=++tim;
    p[tim]=u;
    if(son[fa[u]]!=u)top[u]=u;
    else top[u]=top[fa[u]];
    if(son[u])dfs2(son[u]);
    for(int to:line[u]){
```

```
        if(to==fa[u]||to==son[u])continue;
        dfs2(to);
    }
    return ;
}
```

链上求和:

```
int query(int x,int y){
    int ansn=0;
    while(top[x]!=top[y]){
        if(dep[top[x]]<dep[top[y]])swap(x,y);
        ansn+=getans(1,1,n,dfn[top[x]],dfn[x]);
        //getans()是线段树查[dfn[top[x]],dfn[x]]区间的答案
        ansn%=mod;
        x=fa[top[x]];
    }
    if(dep[x]>dep[y])swap(x,y);
    ansn+=getans(1,1,n,dfn[x],dfn[y]);
    ansn%=mod;
    return ansn;
}
```

常见应用:

树上链查询，子树信息维护和查询。

线段树上查询有个 log ，跳重链有个 log ，查询复杂度一般是 log^2n 的。

还有另一种剖分方式定义重儿子为子树节点中最大深度最大的儿子。

pro

[P3979 遥远的国度]

给定一棵有根树，每个点有一个权值，提供三种操作:
1.将x节点变为根节点
2.将x到y路径上的点的权值全部改为v
3.询问x的子树中点权的最小值

Sol:

我们考虑一下，当前根与点x在原树中的关系无非以下几种:
1.当前根就是点x——那么直接输出整棵树答案就行了。
2.当前根在原树中不在x的子树内——那么哪怕根换成了现在的根，x的子树还是没有变化，那么我们可以直接去原来的子树里查询。
3.当前根在原树中在x的子树内——那么我们设当前根在u点的子树v内，那么所要求的答案就是整棵树除去v这一棵子树的答案。至于怎么求v，可以从root往上跳重链，直到跳到u为止，那么就是看它从哪一个u点的子节点跳过来的就好了。

给定一棵 n 个节点的无根树，共有 m 个操作，操作分为两种：
将节点 a 到节点 b 的路径上的所有点（包括 a 和 b）都染成颜色 c。
询问节点 a 到节点 b 的路径上的颜色段数量。
颜色段的定义是极长的连续相同颜色被认为是一段。例如 112221 由三段组成：11、222、1。

Sol：

跳轻边时判断一下是不是相同颜色，重链上用线段树维护一下。

[P2542 [AHOI2005] 航线规划](#)

给出一张N个点M条边的图，需要支持以下操作

1. 删除 连接u和v的边
2. 询问 u到v的必经边

保证所有删除操作结束后，整个图是联通的
N≤30000 , M≤100000 ，操作数≤40000

Sol：

题目要求支持动态删边，维护每条边是否是桥。

删边不好做，考虑离线下来倒着做，删边变成加边。

考虑特殊情况，初始状态为一棵树。
一开始所有边都是桥。
加上一条边呢？端点之间的树边全都不是桥了

用树剖维护一下就行。

GCD

给出一棵N个节点的树，每个点有权值，需要支持以下操作

1. 将u到v路径上所有点权加v
2. 询问 u到v路径上所有点权的gcd

N≤50000，操作数≤50000

Sol：

跟之前讲的序列上GCD没啥区别，不过是拿到树上了。

[P4175 [CTSC2008] 网络管理](#)

给出一棵树，每个点有点权，需要支持以下操作：

1. 修改某个点的点权
2. 询问 u到v 路径上所有点中，权值第k大的是多少
   数据范围 8w

Sol:

序列上单点修改区间第k大就是树状数组套主席树。

用树剖把这道题搬回序列上就行，复杂度3log但是能过（据说）

考虑每条链可以拆分为 $u + v – lca(u, v) – fa(lca(u, v))$

所以我们可以像序列上那样建出前缀主席树，每个节点的主席树存储的是根到它路径上的所有点，然后同时查询4棵主席树就行了。

然而这样的话修改一个节点的权值会影响到它的子树内所有的节点，于是我们不得不暴力修改O（n）棵主席树。

不过既然一个点的修改只会对子树内的节点产生整体影响，我们可以考虑维护dfs序，因为一棵子树在dfs序上是一段连续的区间。

所以问题就变成了：dfs序上的区间修改，单点查询。这可以用树状数组实现。

复杂度少一个树剖跳重链的log

练习题：

P4211 [LNOI2014] LCA

P5556 圣剑护符 （前置知识：线性基）

CDQ分治

算法概述

用于处理一些离线，修改相互独立，或者如果所有修改都在询问前边会好做的问题。基本套路是，根据某一维分成左右两边，先处理左边，计算左边对右边的贡献，再计算右边

点对计数：找到具有某些特性的点对(i, j)数量
静态变动态：使用log的代价将边修改边询问的问题变成先修改后询问
优化一维DP：DP数组是一维，但是转移条件是多维

- 二维偏序

有n个数对(a_i, b_i)，求出(i, j)的数量，满足 $i \neq j$ 且$a_i \le a_j, b_i \le b_j$

经典方法是按照a从小到大排序，从左往右扫，对于每个j，查询树状数组里[1, bj]的个数，然后将bj添加到树状数组中

还可以用求逆序对的思路，使用归并排序
先按a从小到大排序，对于区间[L, R]，递归处理[L, mid]和[mid+1, R]
计算左区间对右区间的贡献，由于已经排过序了，可以保证左区间的a都不大于右区间，只需要对b归并排序，在合并的过程中就能顺便算出贡献

两种方法时间复杂度均为 $O(nlogn)$

- 三维偏序（CDQ分治）

有 n 个数对 (a_i, b_i, c_i)，求出 (i, j) 的数量，满足 $i \neq j$ 且 $a_i \leq a_j, b_i \leq b_j, c_i \leq c_j$

先按a从小到大排序，再对b归并排序统计答案：
同样的，考虑[L, mid]对[mid+1, R]的贡献，由于已经对a排过序了，所以a可以直接忽略
两区间的b也分别是从小到大排序的，在归并合并的过程中，每次选择左/右区间中b较小的放前面。

每处理一个数：

是左区间的数，就将对应的c加入树状数组。

是右区间的数，查询树状数组中比当前的c小的元素的个数。

由于a先排好序了，再按照b从小到大的顺序处理，只需要看c的相对关系，用树状数组统计。

核心是将区间拆分成两个子区间递归，每次算一个左区间中元素和一个右区间中元素配对的答案

代码实现：

```
void CDQ(int l,int r){
    int mid=(l+r)/2,ll=l,rr=mid+1,nown=l;
    if(mid!=l)CDQ(l,mid),CDQ(mid+1,r);//先递归处理左右子区间，保证子区间按照b从小到大顺序排
    while(ll<=mid&&rr<=r){
        if(point[ll].b<=point[rr].b){//左区间中第一个元素的b小于右区间中第一个元素的b
            addtree(point[ll].c,point[ll].tim);//将左区间中第一个元素的c加入树状数组
            part[nown]=point[ll];ll++;//左区间的第一个元素处理完了，指针ll移向第二个元素
        }
        else{//左区间中第一个元素的b大于右区间中第一个元素的b
            ans[point[rr].num]+=gettree(point[rr].c);//查询左区间中已经处理的元素的c比当前的c小的个数
            part[nown]=point[rr];rr++;
        }
        nown++;
        //part存的是归并后的序列是什么
    }
    while(rr<=r){//左区间中元素处理完了，右区间中还有元素，还要对这些统计一下答案
        ans[point[rr].num]+=gettree(point[rr].c);
        part[nown]=point[rr];rr++;nown++;
    }
    for(int i=l;i<=ll-1;i++)cuttree(point[i].c,point[i].tim);//树状数组只用了一个，要把这次用时的修改删空，防止对下一次的影响
    while(ll<=mid){//左区间元素没用完，要归并一下
        part[nown]=point[ll];
        ll++,nown++;
```

```
    }
    for(int i=l;i<=r;i++)point[i]=part[i];
    return ;
}
```

复杂度树状数组+二分递归 $O(nlog^2n)$

pro

P4390 [BalkanOI2007] Mokia 摩基亚

维护一个W×W的矩阵，初始值均为S. 每次操作可以增加某格子的权值，或询问某子矩阵的总权值

修改次数<= 16w 询问次数<=1w W<=200w

Sol:

差分一下就变成了求(1,1)...(n,m)的矩形和

令时间（t_i）为一维，一共有两个操作：

1. 增加某格子的权值就是在 $(t_i,\ x_i,y_i)$ 上加一个权值
2. 查询 $(1,1)-(x_i,y_i)$的矩阵和就是查询满足 $t_j<t_i,x_j<x_i,y_j<y_i$ 的元素的权值和

原三维偏序统计数对个数在树状数组中加的是1，把1改为权值就行。

拓展：

上面单点加变成矩阵加?

单点加变成矩阵加不影响我们使用CDQ分治，只是统计贡献的方式会发生变化
要求左区间修改对右区间询问的贡献，等价于：先有若干个矩阵加，询问若干个矩阵和
这类问题可使用扫描线算法来做

P2487 [SDOI2011] 拦截导弹

某国为了防御敌国的导弹袭击，发展出一种导弹拦截系统。但是这种导弹拦截系统有一个缺陷：虽然它的第一发炮弹能够到达任意的高度、并且能够拦截任意速度的导弹，但是以后每一发炮弹都不能高于前一发的高度，其拦截的导弹的飞行速度也不能大于前一发。某天，雷达捕捉到敌国的导弹来袭。由于该系统还在试用阶段，所以只有一套系统，因此有可能不能拦截所有的导弹。

在不能拦截所有的导弹的情况下，我们当然要选择使国家损失最小、也就是拦截导弹的数量最多的方案。但是拦截导弹数量的最多的方案有可能有多个，如果有多个最优方案，那么我们会随机选取一个作为最终的拦截导弹行动蓝图。

我方间谍已经获取了所有敌军导弹的高度和速度，你的任务是计算出在执行上述决策时，每枚导弹被拦截掉的概率。

- 对于 100% 的数据，$1\le n\le 5\times 10^4$，$1\le h_i,v_i\le 10^9$。

Sol:

三维上的最长不下降子序列，求长度。

用cdq做DP就行。

看着办：[P4690 [Ynoi2016] 镜中的昆虫]

整体二分

对于单次询问做法是二分答案，但有多次询问时，将询问离线下来。

对于答案值域在 $[l,r]$ 中的询问，判断询问在 $ans=mid=(l+r)/2$ 处是否可行，不可行则放入 答案值域在 $[l,mid]$ 处理，反之放入 $[mid+1,r]$ 处理。

来具体理解一下。

[P1527 [国家集训队] 矩阵乘法]

给你一个 $n\times n$ 的矩阵，不用算矩阵乘法，但是每次询问一个子矩形的第 k 小数。

- 对于 100% 的数据，保证 $1\le n\le 500$，$1\le q\le 6\times 10^4$，$0\le a_{i,j}\le 10^9$。

多个询问的二分答案是可以同时被检验的，我们可以为所有询问同时二分答案，把所有答案小于等于mid的询问放在询问序列的左侧，大于mid的放到询问序列的右侧然后递归处理。

我们每次可以用把矩阵中小于等于mid的元素染成黑色，剩下的元素保持白色。这样对于每一个询问的检验，就相当于是统计某个子矩阵中黑点的个数。为什么不用二维树状数组维护前缀和呢?

最开始的时候整个矩阵都设为0，然后让所有小于等于mid的位置加1。

当向 $[l,mid]$ 递归时，只将数在 $[(l+mid)/2+1,mid]$ 的数对应位置减1。递归完 $[l,mid]$ 后再将这部分加回来。

向 $[mid+1,r]$ 递归时，将数在 $[mid+1,(mid+1+r)/2]$ 的数对应位置加1，递归完后减去。

这样每个数每一层最多被加减一次，一共log层。

每个询问也只会被查询log次，复杂度是二分+二维树状数组三log。

[P3332 [ZJOI2013] K大数查询]

你需要维护 n 个可重整数集，集合的编号从 1 到 n。
这些集合初始都是空集，有 m 个操作：

- `1 l r c`：表示将 c 加入到编号在 $[l,r]$ 内的集合中
- `2 l r c`：表示查询编号在 $[l,r]$ 内的集合的并集中，第 c 大的数是多少。

注意可重集的并是不去除重复元素的，如 $\{1,1,4\}\cup\{5,1,4\}=\{1,1,4,5,1,4\}$。
$1\le n,m\le 5\times 10^4$
$1\le l,r\le n$

Sol:

加一维时间，将修改和询问一起按 c 二分。

若当前值域为 $[l,r]$，按时间依次处理对应的修改和询问并划分。

对 $l<=c<=mid$ 的修改，会对 $[mid+1,r]$ 的询问有影响。

而 $mid+1<=c<=r$ 的修改不会对 $[l,mid]$ 的询问有影响。

和上一题一样的套路。