# 树链剖分：

## 问题：

给你这样一个问题，已知一棵包含N个结点的树（连通且无环），每个节点上包含一个数值，需要支持以下操作：

操作1： 格式： 1 x y z 表示将树从x到y结点最短路径上所有节点的值都加z

操作2： 格式： 2 x y 表示求树从x到y结点最短路径上所有节点的值之和

操作3： 格式： 3 x z 表示将以x为根节点的子树内所有节点值都加上z

操作4： 格式： 4 x 表示求以x为根节点的子树内所有节点值之和

## 解决方案：

这个问题里，操作1和2都不能仅仅用线段树实现，树链剖分：就是对一棵树分成几条链，把树形变为线性，减少处理难度，定义如下概念：

* 重结点：子树结点数目最多的结点；
* 轻节点：父亲节点中除了重结点以外的结点都是轻节点
* 重边：父亲结点和重结点连成的边；
* 轻边：父亲节点和轻节点连成的边；
* 重链：由多条重边连接而成的路径；
* 轻链：由多条轻边连接而成的路径；

参考图片里，树黑边是重边，红点是重节点，可见除了根，任何轻节点的的父节点一定在重链上，并且重链顶端的节点的父节点是另外一条重链

需要维护数组有：

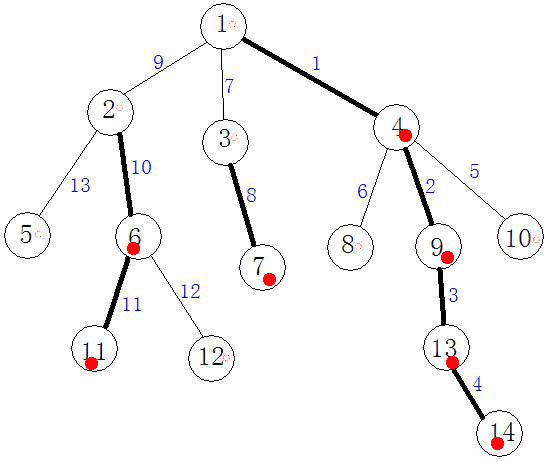
* h[u]代表u的深度
* p[u]代表u的父节点编号
* sz[u]代表子树中，节点总个数
* top[u]代表节点 u所在重链的最顶节点编号
* shv记录某个节点子节点中的重节点编号，或者isheavy数组记录某个节点是不是重节点。二者皆可。网上大多采取shv的方式，因为我试验这种方法稍微快一点

初始化top,每个节点top[u]=u，也就是等于自身，两次dfs遍历树，需要进行两次的dfs，第一次标记p,h,sz,shv，第二次找轻重边和轻重链子。也就是标记top

第一步：对树进行第一次dfs1做4件事：标记每个点深度h[],标记每个点的父亲p[],标记每个非叶子节点的子树大小(节点个数), 标记每个非叶子节点的重儿子编号shv[]

第二步：对树进行第二次dfs2做2件事：对每个点新编号id[i]代表原来编号i的新编号；top[i]表示每个点所在重链顶端的节点编号，注意这里一定要先处理重儿子再处理轻儿子。

参考图片：



dfs2的顺序是先处理重儿子再处理轻儿子，这保证了：因为顺序是先重再轻，所以每一条重链的新编号是连续的；因为是dfs，所以每一个子树的新编号也是连续的。来回顾问题：

**操作2：**求树从x到y结点最短路径上所有节点的值之

，设两个点x和y较深的是deep(x,y),每次选取x和y中top[x]和top[y]较深的点，执行如下操作：deep(x,y)是重链上的节点，则跳到这条重链顶节点的父节点，deep(x,y)不是重链节点，就跳到父节点。执行这个操作直到x和y在同一条重链上(通过top判定)，每次跳链时候要计算经过的点权之和s。sum(x,y)表示同一条重链下的x和y经过的点的权值和，s+sum(x,y)就是答案.其中s实际上也是通过若干个sum(x,y)求得的

怎么求和呢，肯定不可以一个个加那样就是暴力，这是新标号数组id就用上了，同一个重链下id是连续的，那么就可以用线段树或者树状数组这类区间查询结构，维护每个重链的值，建1个线段树即可，既可以维护每条重链的区间和，又可以维护操作4求子树的和。时间复杂度是logn\*logn

**操作1：**将树从x到y结点最短路径上所有节点的值都加z，类似的就是线段树区间修改操作，和类似刚才说的方法。

**操作4：**求以x为根节点的子树内所有节点值之和，

由于按照dfs序建立的新id, 实际上某个节点子节点一定是连续的(看新id)，直接查询区间和就好。复杂度logn

**操作3：**以x为根节点的子树内所有节点值都加上z

和操作4类似方法，就是线段树区间修改。