到目前为止，我们用数据结构处理的大多是序列上的问题。这些问题的形式一般是给定序列中的两个位置 l 和 r ，在区间 $[l, r]$ 上执行查询或修改指令。如果给定一棵树，以及树上两个节点 x 和 y，那么与 "序列上的区间" 相对应的就是 "树上两点之间的路径"。先不考虑对路径进行修改的操作。本文中介绍的点分治就是在一棵树上，**对具有某些限定条件的路径静态地进行统计**的算法。

【例题】POJ-1741 Tree

题目描述

给一棵有 n 个点的树，每条边都有一个权值（小于 1001）。树上两个节点 x 与 y 之间的路径长度就是路径上各条边的权值之和。求长度不超过 k 的路径有多少条($n \leq 10000$)。

分析

本题树中的边是无向的，即这棵树是一个由 n 个点、$n-1$ 条边构成的无向连通图。我们把这种树称为"无根树"，也就是说可以任意指定一个节点为根节点,而不影响问题的答案。

若指定节点 p 为根，则对 p 而言，树上的路径可以分为两类：

1.经过根节点 p。

2.包含于 p 的某一棵子树中（不经过根节点）。

根据分治的思想，对于第 2 类路径，显然可以把 p 的每棵子树作为子问题，递归进行处理。

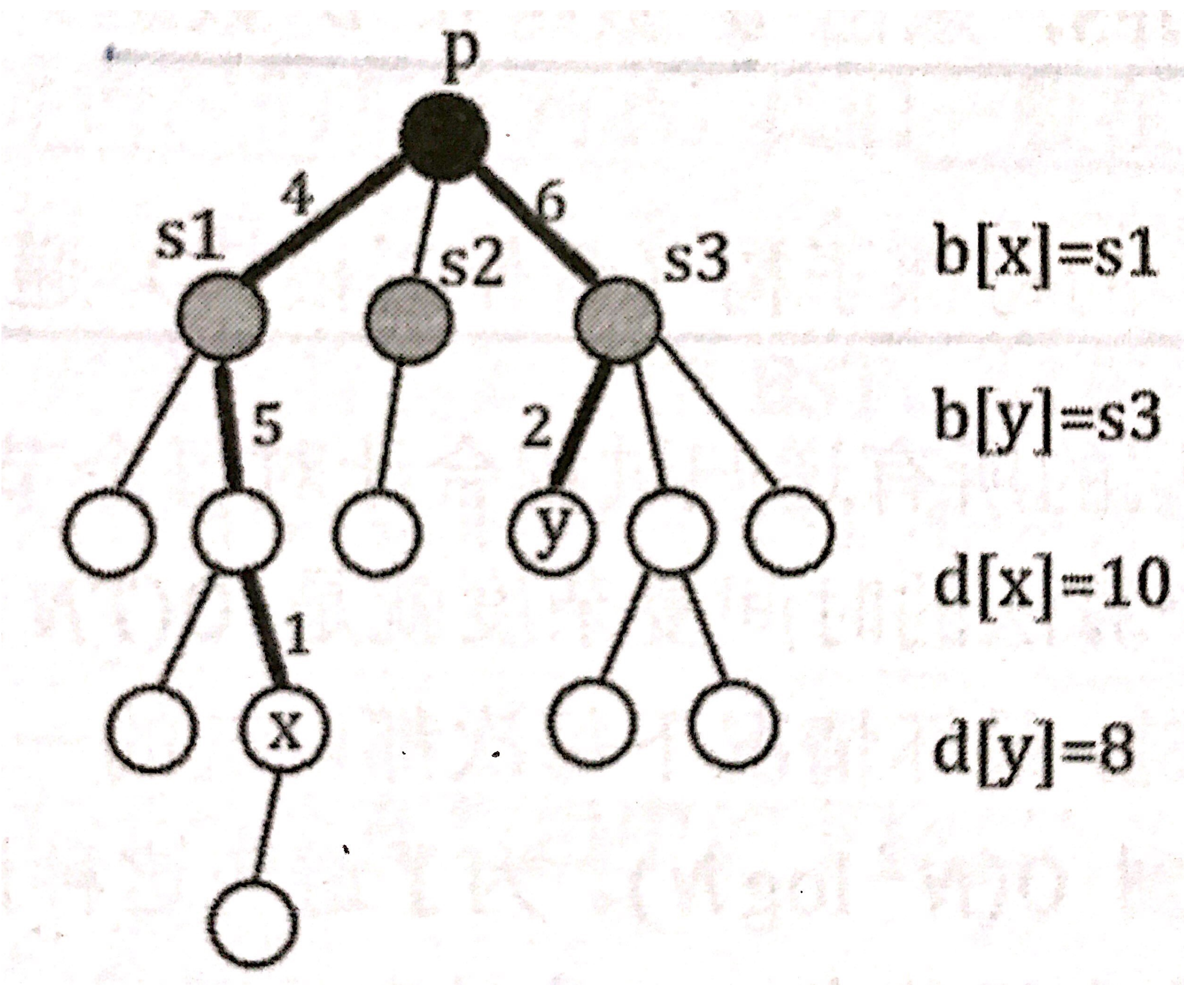
而对于第 1 类路径，可以从根节点 p 分成 "x ~ p" 与 "p ~ y" 两段。我们可以从 p 出发对整棵树进行 DFS ，求出数组 d，其中 $d[x]$ **表示点 x 到根节点 p 的距离**。同时还可以求出数组 b，其中 $b[x]$ **表示点 x 属于根节点 p 的哪一棵子树**，特别地，令 $b[p] = p$。

此时满足题目要求的第 1 类路径就是满足以下两个条件的**点对** (x, y) 的个数：

1.$b[x] \neq b[y]$。

2.$d[x] + d[y] \leq k$。

如下图所示。



定义 Calc(p) 表示在以 p 为根的树中统计上述点对的个数（第 1 类路径的条数）。Cal(p) 有两种常见的实现方式。针对不同的题目,二者各有优劣。

方法一：树上直接统计

设 p 的子树为 $s_1, s_2, \cdots, s_m$。

对于 s_i 中的每个节点 x，把在子树 $s_1, s_2, \cdots, s_i - 1$ 中的满足 $d[x] + d[y] \le k$ 的节点 y 的个数累加到答案中即可。

具体来说，可以建立一个树状数组，依次处理每棵子树 s_i。

1. 对 s_i 中的每个节点 x，查询前缀和 $\mathrm{ask}(k - d[x])$，即为所求的 y 的个数。

2. 对 s_i 中的每个节点 x，执行 $\mathrm{add}(d[x], 1)$，表示与 p 距离为 $d[x]$ 的节点增加了 1 个。

按子树一棵棵进行处理保证了 $b[x] \ne b[y]$，查询前缀和保证了 $d[x] + d[y] \le k$。

需要注意的是,树状数组的范围与路径长度有关，这个范围远比 n 要大。而本题中不易进行离散化。一种解决方案是用平衡树代替树状数组，以保证 $O(n \log n)$ 的复杂度,但代码复杂程度显著增加。所以本题更适用下一种方法。

方法二：指针扫描数组

把树中每个点放进一个数组 a，并把数组 a 按照节点的 d 值排序。

使用两个指针 L, R 分别从前、后开始扫描 a 数组。

容易发现，在指针 L 从左向右扫描的过程中，恰好使得 $d[a[L]] + d[a[R]] \le k$ 的指针 R 的范围是从右向左单调递减的。

另外，我们用数组 $cnt[s]$ 维护在 $L + 1$ 与 R 之间满足 $b[a[i]] = s$ 的位置 i 的个数。

于是，当路径的一端 x 等于 $a[L]$ 时，满足题目要求的路径另 一端 y 的个数就是 $R-L-cnt[b[a[L]]]$。 总而言之，整个点分治算法的过程就是：

1. 任选一个根节点 p （后面我们将说明，p 应该取树的重心）。

2. 从 p 出发进行一次 DFS，求出 d 数组和 b 数组。

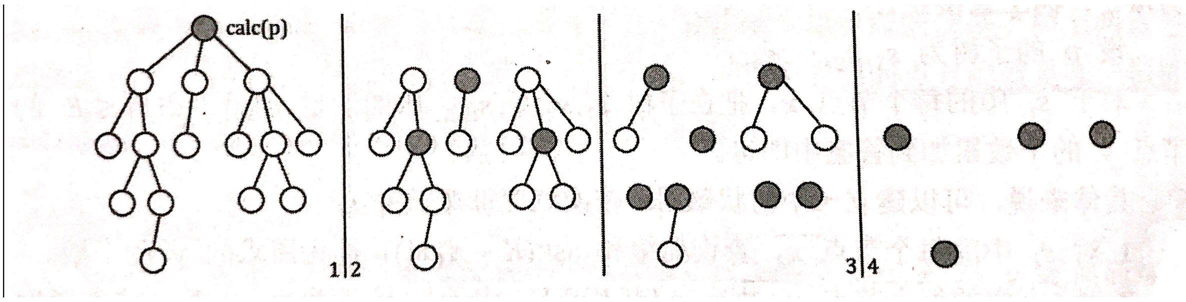
3. 执行 $\mathrm{Calc}(p)$。

4.删除根节点p，对 p 的每棵子树(看作无根树)递归执行 $1\sim4$ 步。

在点分治过程中，每一层的所有递归过程合计对每个节点处理 1 次。因此，若递归最深到达第 T 层，则整个算法的时间复杂度就是 $O(Tn\log n)$。

如果问题中的树是一条链，最坏情况下每次都以链的一端为根，那么点分治将需要递归 n 层，时间复杂度退化到 $O(n^2\log n)$。为了避免这种情况，我们**每次选择树的重心作为根节点** p。此时容易证明 p 的每棵子树的大小都不会超过整棵树大小的一半，点分治就至多递归 $O(\log n)$ 层，整个算法的时间复杂度为 $O(n\log^2 n)$。

如下图所示。



代码

```
#include<iostream>
#include<cstdio>
#include<cstring>
#include<algorithm>
using namespace std;
const int SIZE=100010;
const int INF=0x3f3f3f3f;
struct Edge
{
    int Next;
    int to;
    int dis;
}edge[SIZE<<1];
int head[SIZE],d[SIZE],size[SIZE],temp[SIZE];
int num_edge,num,ans,root,sz;
int now_size=0x3f3f3f3f;
int n,k;
bool vis[SIZE];
void add_edge(int from,int to,int dis)
{
    edge[++num_edge].to=to;
    edge[num_edge].dis=dis;
    edge[num_edge].Next=head[from];
    head[from]=num_edge;
}
void dfs_root(int x,int fa)//求树的重心
{
```

```
    size[x]=1;
    int max_part=0;
    for(int i=head[x];i!=-1;i=edge[i].Next)
    {
        int y=edge[i].to;
        if(vis[y]||fa==y)
            continue;
        dfs_root(y,x);
        size[x]=size[x]+size[y];
        max_part=max(max_part,size[y]);
    }
    max_part=max(max_part,sz-size[x]);
    if(now_size>max_part)
    {
        now_size=max_part;//全局变量now_sz记录了重心对应的max_part
        root=x;//全局变量pos记录了重心
    }
}
void get_dis(int x,int fa)//求出d数组
{
    temp[++num]=d[x];
    for(int i=head[x];i!=-1;i=edge[i].Next)
    {
        int y=edge[i].to;
        if(vis[y]||fa==y)
            continue;
        d[y]=d[x]+edge[i].dis;
        get_dis(y,x);
    }
}
int calc(int x,int dis)//计算符合要求的点对的个数
{
    d[x]=dis;
    num=0;
    get_dis(x,x);
    sort(temp+1,temp+num+1);
    int l=1,r=num,ans=0;
    while(l<r)
    {
        while(temp[l]+temp[r]>k)
            r--;
        if(l<r)
            ans=ans+(r-l);
        l++;
    }
    return ans;
}
void dfs(int x)
{
    vis[x]=true;
    ans=ans+calc(x,0);
    for(int i=head[x];i!=-1;i=edge[i].Next)
    {
        int y=edge[i].to;
        if(vis[y])
            continue;//点y已经被访问过了
        ans=ans-calc(y,edge[i].dis);
        now_size=INF;
```

```
        root=0;
        sz=size[y];
        dfs_root(y,0);
        dfs(root);
    }
}
int main()
{
    while(cin>>n>>k&&n&&k)
    {
        memset(vis,false,sizeof(vis));
        memset(head,-1,sizeof(head));
        ans=0;
        num_edge=0;
        for(int i=1;i<=n-1;i++)
        {
            int u,v,w;
            scanf("%d %d %d",&u,&v,&w);
            add_edge(u,v,w);
            add_edge(v,u,w);
        }
        dfs(1);
        cout<<ans<<endl;
    }
    return 0;
}
```