Problem K. Kazusa!!

Input file:	standard input	Time limit:	1 second
Output file:	standard output	Memory limit:	256 megabytes

冬马并不想理你并向你扔了一道阅读理解。

一道2019年新生赛试题是阅读理解题当且仅当这道题的题解被写在了题面里。

子树的概念

设 T 是有根树，a 是 T 中的一个顶点，由 a 以及 a 的所有子孙导出的子图称为树 T 的子树。

显然，a 的子树的顶点集是由所有顶点到根的最短路上经过 a 的顶点构成的。

如何存储一颗含有n个顶点的树

```
#include <bits/stdc++.h>
using namespace std;

const int N=1e6+5;
int n;
vector<int> e[N];

int main()
{
    scanf("%d",&n);
    for(int i=0,x,y;i<n;i++)
        scanf("%d%d",&x,&y),e[x].push_back(y),e[y].push_back(x);
    return 0;
}
```

以上代码实现了树的读入与存储，其中 $e[i]$ 表示与 i 号顶点相邻的所有顶点，也就是说你可以通过访问 $e[i][j]$ 来获得 i 号顶点的某个相邻顶点。

每读入一条无向边 $x \leftrightarrow y$ 就把它加入 e 中。

如何遍历一颗树

```
void dfs(int now,int fa)
{
    print("now: %d\n",now);
    for(int i=0;i<e[now].size();i++)
        if(e[now][i]!=fa) dfs(e[now][i],now);
    /*
    你也可以这样写
    for(auto to:e[now])
        if(to!=fa) dfs(to,now);
    */
}

// 主函数里调用dfs(1,1)
```

以上代码实现了以 1 为根深度优先遍历整棵树，可以手玩一下加深对遍历过程的理解。

现在我们来解决以下问题

给定一颗包含 n 个顶点且以 1 为根的有根树，树的每个顶点都有一个数字。我们称数字 c 是顶点 v 的子树中的众数当且仅当没有其他数字在顶点 v 的子树中的出现次数严格大于 c 的出现次数。显然，可能同时存在多个数字是一颗子树的众数。请计算出对于每一个顶点的子树中，所有众数之和。

一种众数在一颗子树内只会贡献一次。

一个朴素的 $O(n^2)$ 的做法

```
void calc(int now,int fa)
{
    cnt[c[now]]+=y;
    if(cnt[c[now]]==mx) sum+=c[now];
    else if(cnt[c[now]]>mx) mx=cnt[c[now]],sum=c[now];
    for(auto to:e[now])
        if(to!=fa) cal(to,now,y);
}

void dfs(int now,int fa)
{
    for(auto to:e[now])
        if(to!=fa) dfs(to,now);
    calc(now,fa);// 统计这棵子树的信息
    ans[now]=sum;
    sum=0,mx=0;
}
```

其实就是枚举树上的每个点，扫一次子树得到答案。

显然这样做太慢了，深度比较大的顶点被反复统计了多次，我们有没有办法重用这部分信息?

dsu on tree

有人称作"树上启发式合并"，也有人称作"边分治"，事实上它是一种猩猩也能懂的优雅的暴力。

暴力的代码中，当我们处理完当前顶点的所有儿子的信息的时候，我们会调用当前顶点的 $calc$ 来统计当前这棵子树的信息，相当于要重新访问这棵子树内的所有顶点。注意到回溯上来的时候我们可以保存其中某个儿子的子树的信息，这样我们就可以少访问一部分子孙顶点了。

问题变成了保留哪个儿子。我们想要保留尽可能多的顶点的信息，选择子树大小最大的儿子就行了，这样的儿子我们称为重儿子。一个顶点和它重儿子之间的边称为重边，反之称为轻边。重边连起来得到的链称为重链。

整个算法步骤如下:

1. 先做一次重链剖分，得到每个顶点的重儿子。
2. dfs 遍历整棵树。先遍历轻儿子统计完轻儿子的答案，再遍历重儿子的答案。
3. 如果当前顶点不是重儿子则清除子树的信息。

详见代码(仅供参考)。

```
void cal(int now,int fa,int y)
{
    // y==1表示统计 y==-1表示清除
    cnt[c[now]]+=y;
    if(cnt[c[now]]==mx) sum+=c[now];
    else if(cnt[c[now]]>mx) mx=cnt[c[now]],sum=c[now];
    for(auto to:e[now])
        if(to!=fa&&to!=hson) cal(to,now,y);
}

void dfs2(int now,int fa,int keep)
{
    // keep==1 表示这是一个重儿子 否则就是一个轻儿子
    for(auto to:e[now])
    {
        if(to==fa||to==son[now]) continue;
        dfs2(to,now,0);
    }
    if(son[now]) dfs2(son[now],now,1); // 如果有重儿子则遍历重儿子
    hson=son[now];
    cal(now,fa,1); // 遍历非重儿子的子树
    hson=0;
    ans[now]=sum;
    if(!keep) cal(now,fa,-1),sum=0,mx=0; // 如果是轻儿子则清除统计来的信息
}
```

时间复杂度分析

$Lemma$: 设 $size(x)$ 表示 x 为根的子树的大小。对于一条轻边 (x, y) ，其中 x 是 y 的父亲，则有 $2 \times size(y) < size(x)$ 。

证明: 因为 y 是轻儿子，则存在 x 的重儿子 z 满足 $size(z) > size(y)$ 。因为 $size(x) \geq size(z) + size(y) + 1$ ，带入得 $size(x) > size(y) + size(y) + 1$ 。

这意味着每经过一条轻边，子树大小至少翻一番。换而言之，任何一个顶点到根的路径上轻边的个数是 $O(log_2 n)$ 的。一个顶点会被重复统计当且仅当它在某个轻儿子的子树中，由于链上轻儿子的个数是 $O(log_2 n)$ 的，所以它只会被重复计算 $O(log_2 n)$ 次，总时间复杂度 $O(nlogn)$ 。

Input

第一行一个正整数 $n(1 \leq n \leq 10^5)$ 。

接下来一行 n 个整数 $c_i(1 \leq c_i \leq n)$，表示第 i 个顶点上的权值。

接下来 $n-1$ 行，每行两个整数 $x, y(1 \leq x, y \leq n)$ ，表示 x 与 y 之间有一条边。

Output

输出一行 n 个整数，其中第 i 个数表示以 i 为根的子树中所有众数的和。

Samples

standard input	standard output
4 1 2 3 4 1 2 2 3 2 4	10 9 3 4
15 1 2 3 1 2 3 3 1 1 3 2 2 1 2 3 1 2 1 3 1 4 1 14 1 15 2 5 2 6 2 7 3 8 3 9 3 10 4 11 4 12 4 13	6 5 4 3 2 3 3 1 1 3 2 2 1 2 3