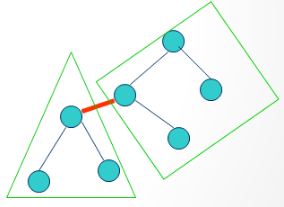
# 边分治学习笔记（bzoj2870)

### Descintption

　　H城很大，有N个路口（从1到N编号），路口之间有N-1边，使得任意两个路口都能互相到达，这些道路的长度我们视作一样。每个路口都有很多车辆来往，所以每个路口i都有一个拥挤程度v[i]，我们认为从路口s走到路口t的痛苦程度为s到t的路径上拥挤程度的最小值，乘上这条路径上的路口个数所得的积。现在请你求出痛苦程度最大的一条路径，你只需输出这个痛苦程度。

### Simple Descintption

　　给定一棵N个点的树，求树上一条链使得链的长度乘链上所有点中的最小权值所得的积最大。其中链长度定义为链上点的个数。

### Input

　　第一行N

　　第二行N个数分别表示1~N的点权v[i]

　　接下来N-1行每行两个数x、y，表示一条连接x和y的边

### Output

　　一个数，表示最大的痛苦程度。

### Sample Input

|  |  |
| --- | --- |
| 1  2  3  4 | 3  5 3 5  1 2  1 3 |

### Sample Output

|  |  |
| --- | --- |
| 1 |  |

### 样例解释

　　选择从1到3的路径，痛苦程度为min(5,5)\*2=10

### HINT

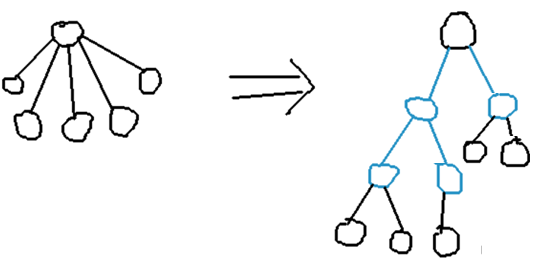
　　100%的数据n<=50000

　　其中有20%的数据树退化成一条链

　　所有数据点权<=65536

Hint：建议答案使用64位整型

如果点分治的话，那个取最小点权操作很难合并，不过树可以点分治，那自然也可以边分治。我们每次找到一条边，使得删掉这条边后的两个连通块中最大的那个的大小尽可能小，然后处理经过这条边的路径信息……   
**桥豆麻袋，这个复杂度好像不太对啊。**   
**在菊花图上，好像可以轻易变成**O(n)**的啊！！！**   
于是我们就要把树做一下转化，如果一个点有太多个儿子，我们就建立若干个虚拟点来管理它的儿子们，这样这棵树就变成了二叉树，可以保证O(nlogn)



我们发现，只要把虚拟点的点权设为原来那个父亲的点权，就不会影响找点权最小值的操作。但是找链条的长度的话，如果两个点的LCA是一个虚拟点的话，它们真实的父亲的贡献就不会被计算。所以我们令真实边的边权为1，虚拟边边权为0，那么一条链的长度就是这条链边权和+1。

这样就好办多了，每次我们找到一条边，使得删掉这条边后的两个连通块中最大的那个的大小尽可能小，然后把边两端的路径存下来，按照点权最小值排序，然后对于A端的一条路径，我们维护B端点权大于等于该路径的路径中，长度最长的一条，对B端也这么做一边就可以得到答案。复杂度O(nlog2n) 。

至于空间，我们知道线段树要开四倍空间，所以对于菊花图，这个重建树就也是四倍空间。

#include<bits/stdc++.h>

using namespace std;

int read() {

int q=0;char ch=' ';

while(ch<'0'||ch>'9') ch=getchar();

while(ch>='0'&&ch<='9') q=q\*10+ch-'0',ch=getchar();

return q;

}

typedef long long LL;

const int N=200005,inf=0x3f3f3f3f;

int n,kn,tot,rt,mxx;LL ans;

int v[N],sz[N],h[N],ne[N<<1],to[N<<1],d[N],vis[N],js[2];

struct node{int v,len;}t[2][N];

bool cmp(node x,node y) {return x.v>y.v;}

vector<int> a[N];

void add(int x,int y,int z) {to[++tot]=y,ne[tot]=h[x],h[x]=tot,d[tot]=z;}

void dfs1(int x,int las) {//建初树

for(int i=h[x];i;i=ne[i])

if(to[i]!=las) a[x].push\_back(to[i]),dfs1(to[i],x);

}

void rebuild() {//重新建树

tot=1;for(int i=1;i<=n;++i) h[i]=0;

for(int i=1;i<=n;++i) {

int sz=a[i].size();

if(sz<=2) {

for(int j=0;j<sz;++j)

add(i,a[i][j],(a[i][j]<=kn)),add(a[i][j],i,(a[i][j]<=kn));

}

else {

int o1=++n,o2=++n;v[o1]=v[o2]=v[i];

add(i,o1,0),add(o1,i,0),add(i,o2,0),add(o2,i,0);

for(int j=0;j<sz;++j)

if(j&1) a[o2].push\_back(a[i][j]);

else a[o1].push\_back(a[i][j]);

}

}

}

void getrt(int x,int las,int SZ) {//找出中心边

sz[x]=1;

for(int i=h[x];i;i=ne[i]) {

if(vis[i>>1]||to[i]==las) continue;

getrt(to[i],x,SZ),sz[x]+=sz[to[i]];

int kl=max(sz[to[i]],SZ-sz[to[i]]);

if(kl<mxx) mxx=kl,rt=i;

}

}

void dfs2(int o,int x,int las,int len,int val) {//计算路口的最小及连长

val=min(val,v[x]),t[o][++js[o]]=(node){val,len};

for(int i=h[x];i;i=ne[i]) {

if(vis[i>>1]||to[i]==las) continue;

dfs2(o,to[i],x,len+d[i],val);

}

}

void work(int x,int SZ) {//在树x中找最大的痛苦程度

mxx=inf,getrt(x,0,SZ);//找中心边rt

if(mxx==inf) return;

int now=rt;vis[now>>1]=1;//断边（双向边原因，2i、2i+1对应i边），或vis[now]=1,vis[now^1]=1

js[0]=js[1]=0,dfs2(0,to[now],0,0,inf),dfs2(1,to[now^1],0,0,inf);

sort(t[0]+1,t[0]+js[0]+1,cmp),sort(t[1]+1,t[1]+js[1]+1,cmp);

for(int i=1,j=1,mxl=0;i<=js[0];++i) {

while(j<=js[1]&&t[1][j].v>=t[0][i].v) mxl=max(mxl,t[1][j].len),++j;

//i到j的路径上拥挤程度的最小值，乘上这条路径上的路口个数所得的积

if(j!=1) ans=max(ans,1LL\*t[0][i].v\*(mxl+t[0][i].len+d[now]+1));

}

for(int i=1,j=1,mxl=0;i<=js[1];++i) {

while(j<=js[0]&&t[0][j].v>=t[1][i].v) mxl=max(mxl,t[0][j].len),++j;

if(j!=1) ans=max(ans,1LL\*t[1][i].v\*(mxl+t[1][i].len+d[now]+1));

//j!=1，至少有两个点（路口）；+1，路口数比边数多1。

}

int ksz=sz[to[now]];

work(to[now],ksz),work(to[now^1],SZ-ksz);

}

int main(){

int x,y;

n=kn=read();

for(int i=1;i<=n;++i) v[i]=read();

for(int i=1;i<n;++i) x=read(),y=read(),add(x,y,1),add(y,x,1);

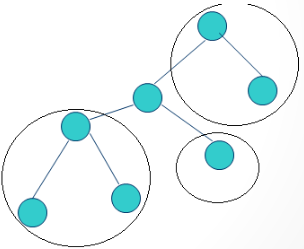
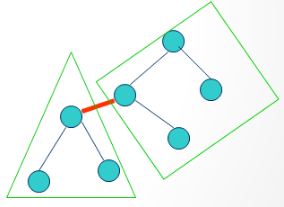
dfs1(1,0),rebuild(),work(1,n);

printf("%lld",ans);

return 0;

}

**树点分治、边分治总结**



分治，把一个复杂的问题，分解为若干（独立或关联）子问题（归并排序、快排、CDQ分治、RMQ、数状数组、线段树，分块，树型DP）

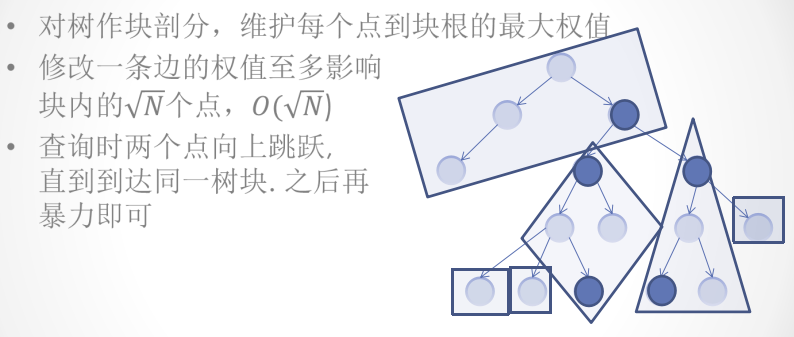
点对（i,j,i<j)：单调性

双条件：枚举其中一个条件（单调），统计另一个

梅花图：二叉树

断开双向边：2，31 4，52 6，7 3 ……

# **[树上分块](https://www.cnblogs.com/hua-dong/p/8275227.html)**



栈：top-now>=，每一块是树上的连通块。

