虚树

算法简介

一种用于加速树上 dp 算法，时间复杂度 $O(k\log k)$ 其中 k 为树上的关键节点数。

算法思想

树上关键点数量较少时很多节点不需要进行 dp 考虑重建一棵树，只保留必要的节点。

发现所有关键节点的 LCA 必须保留，但如果暴力枚举每对关键节点的 LCA 显然会超时。

事实上只需要将关键点序列dfs 序排序，然后枚举序列中相邻节点的 LCA

可以证明该方法枚举得到的 LCA 不会有遗漏。假设 $p=\text{LCA}(u,v)$ 且 u,v 不是序列中相邻的节点。

于是必有 u,v 属于 p 的不同子树，考虑取序列中与 v 相邻且 dfs 序小于 v 的节点，记为 v_2

首先易知 v_2 也位于 p 的子树。于是如果 v_2 与 v 不在同一棵子树，那么 $\text{LCA}(v_2,v)=p$

否则把 v 换成 v_2 继续重复上述操作，最后总有 v_{k-1},v_k 不在同一棵子树(最差的情况是 $v_k=u$) 证毕。

然后为了保证最终选取的点一定会构成树而不是森林，考虑强制将根节点加入虚树或者构造一个虚根。

接下来是建边过程，考虑用单调栈维护从根节点出发的一条树链。

每新加入一个节点时，先计算新节点与栈顶节点(事实上栈顶节点一定是上一次加入的节点，即与新元素相邻的节点)的 LCA 记为 p

如果 p 恰好为栈顶节点，则直接将新节点入栈。

否则，将栈顶节点弹出直到栈顶节点的下一个节点的 dfs 序不大于 p 的dfs 序。

注意到每当栈顶节点被弹出时他在虚树中的位置已经确定，可以直接将他与下一个栈顶节点连一条边。

然后如果 p 恰为栈顶栈顶节点的下一个节点，则将栈顶节点弹出并将他与下一个栈顶节点连一条边。

否则将栈顶节点弹出并将他与 p 连一条边，再将 p 加入栈。这一系列操作结束后再将新节点入栈。

所有关键节点都访问结束后将栈的剩余元素出栈并连边。

算法模板

洛谷p2495

题意

给定一棵以 1 为根的边权树，设切断一条边的费用为这条边的边权。接下来 q 次询问。

每次询问给定 k_i 个关键节点(保证不含根节点)，要求切断若干条边使得所有关键节点节点与根节点不连通，问该操作的最小费用为多少。

题解

首先考虑 dp 过程，有状态转移方程

\begin{equation}v \text{ 是 }u \text{的儿子节点且} v \text{ 不是关键节点}\text{dp}(u)\gets\min(\text{dp}(v),w(u,v))\end{equation} \begin{equation}v \text{ 是 }u \text{的儿子节点且} v \text{ 是关键节点}\text{dp}(u)\gets w(u,v)\end{equation}

接下来建立虚树，并令 $w(u,v)$ 为 $u\to v$ 路径上的最短边。事实上令 $w(u,v)$ 为根节点 $\to v$ 的最短边不影响最终答案。

最后暴力 dp 即可，时间复杂度 $O(\sum_{i=1}^q k_i\log k_i)$注意每次新建树时不能暴力清零 head 数组，需要在建树过程中清零。

```
const int MAXN=3e5+5;
struct Edge{
    int to,w,next;
}edge[MAXN<<1];
int head[MAXN],edge_cnt;
void Insert(int u,int v,int w){
    edge[++edge_cnt]=Edge{v,w,head[u]};
    head[u]=edge_cnt;
}
int d[MAXN],dis[MAXN],sz[MAXN],f[MAXN],dfn[MAXN],dfs_t;
int h_son[MAXN],mson[MAXN],p[MAXN];
void dfs_1(int u,int fa,int depth,int Dis){
    sz[u]=1;f[u]=fa;d[u]=depth;mson[u]=0;dfn[u]=++dfs_t;dis[u]=Dis;
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(v==fa)
        continue;
        dfs_1(v,u,depth+1,min(Dis,edge[i].w));
        sz[u]+=sz[v];
        if(sz[v]>mson[u])
        h_son[u]=v,mson[u]=sz[v];
    }
}
void dfs_2(int u,int top){
```

```
    p[u]=top;
    if(mson[u])dfs_2(h_son[u],top);
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(v==f[u]||v==h_son[u])
        continue;
        dfs_2(v,v);
    }
}
int LCA(int u,int v){
    while(p[u]!=p[v]){
        if(d[p[u]]<d[p[v]])swap(u,v);
        u=f[p[u]];
    }
    return d[u]<d[v]?u:v;
}
struct cmp{
    bool operator () (const int a,const int b)const{
        return dfn[a]<dfn[b];
    }
};
int Stack[MAXN],top,node[MAXN];
void build(int k){
    edge_cnt=0;
    sort(node,node+k,cmp());
    Stack[top=1]=1;head[1]=0;
    _for(i,0,k){
        int p=LCA(Stack[top],node[i]);
        if(p!=Stack[top]){
while(dfn[p]<dfn[Stack[top-1]])Insert(Stack[top-1],Stack[top],dis[Stack[top
]]),top--;
            if(p!=Stack[top-1])
            head[p]=0,Insert(p,Stack[top],dis[Stack[top]]),Stack[top]=p;
            else
            Insert(Stack[top-1],Stack[top],dis[Stack[top]]),top--;
        }
        head[node[i]]=0,Stack[++top]=node[i];
    }
    while(top>1)Insert(Stack[top-1],Stack[top],dis[Stack[top]]),top--;
}
bool is_key[MAXN];
LL dp(int u){
    LL ans=0;
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(is_key[v])
        ans+=edge[i].w;
        else
        ans+=min(dp(v),1LL*edge[i].w);
    }
    return ans;
```

```
}
int main()
{
    int n=read_int(),root=1,u,v,w;
    _for(i,1,n){
        u=read_int(),v=read_int(),w=read_int();
        Insert(u,v,w);Insert(v,u,w);
    }
    dfs_1(root,0,0,1e9);
    dfs_2(root,root);
    int q=read_int();
    while(q--){
        int k=read_int();
        _for(i,0,k){
            node[i]=read_int();
            is_key[node[i]]=true;
        }
        build(k);
        enter(dp(root));
        _for(i,0,k)
        is_key[node[i]]=false;
    }
    return 0;
}
```

算法练习

习题一

洛谷p4103

题意

给定一棵 n 个节点的无根树，q 次询问。

每次询问给定 k_i 个关键点，要求输出所有关键节点间的路径长度的和、最小值、最大值。

题解

对所有关键节点间的路径长度的和，可以考虑每条虚树上边的贡献，该贡献等于 $w(u,v)\times sz(v)\times (k_i-sz(v))$。

对路径长度的最小值和最大值，可以依次枚举每棵子树中的最短/最长路径，然后考虑与之前枚举的最小值和最大值合并。

```
const int MAXN=1e6+5;
struct Edge{
    int to,w,next;
}edge[MAXN<<1];
int head[MAXN],edge_cnt;
void Insert(int u,int v,int w=1){
    edge[++edge_cnt]=Edge{v,w,head[u]};
    head[u]=edge_cnt;
}
int d[MAXN],sz[MAXN],f[MAXN],dfn[MAXN],dfs_t;
int h_son[MAXN],mson[MAXN],p[MAXN];
void dfs_1(int u,int fa,int depth){
    sz[u]=1;f[u]=fa;d[u]=depth;mson[u]=0;dfn[u]=++dfs_t;
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(v==fa)
        continue;
        dfs_1(v,u,depth+1);
        sz[u]+=sz[v];
        if(sz[v]>mson[u])
        h_son[u]=v,mson[u]=sz[v];
    }
}
void dfs_2(int u,int top){
    p[u]=top;
    if(mson[u])dfs_2(h_son[u],top);
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(v==f[u]||v==h_son[u])
        continue;
        dfs_2(v,v);
    }
}
int LCA(int u,int v){
    while(p[u]!=p[v]){
        if(d[p[u]]<d[p[v]])swap(u,v);
        u=f[p[u]];
    }
    return d[u]<d[v]?u:v;
}
struct cmp{
    bool operator () (const int a,const int b)const{
        return dfn[a]<dfn[b];
    }
};
int Stack[MAXN],top,node[MAXN];
void build(int k){
    edge_cnt=0;
    sort(node,node+k,cmp());
    Stack[top=1]=1;head[1]=0;
```

```
    _for(i,0,k){
        if(node[i]==1)continue;
        int p=LCA(Stack[top],node[i]);
        if(p!=Stack[top]){
while(dfn[p]<dfn[Stack[top-1]])Insert(Stack[top-1],Stack[top],d[Stack[top]]
-d[Stack[top-1]]),top--;
            if(p!=Stack[top-1])
            head[p]=0,Insert(p,Stack[top],d[Stack[top]]-d[p]),Stack[top]=p;
            else
            Insert(Stack[top-1],Stack[top],d[Stack[top]]-
d[Stack[top-1]]),top--;
        }
        head[node[i]]=0,Stack[++top]=node[i];
    }
    while(top>1)Insert(Stack[top-1],Stack[top],d[Stack[top]]-
d[Stack[top-1]]),top--;
}
bool is_key[MAXN];
int tot,cnt[MAXN],dis[MAXN][2],ans2,ans3;LL ans1;
void dp(int u){
    cnt[u]=is_key[u],dis[u][0]=is_key[u]?0:MAXN,dis[u][1]=0;
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        dp(v);
        ans1+=1LL*cnt[v]*(tot-cnt[v])*edge[i].w;
        if(cnt[u]){
            ans2=min(ans2,dis[u][0]+dis[v][0]+edge[i].w);
            ans3=max(ans3,dis[u][1]+dis[v][1]+edge[i].w);
        }
        dis[u][0]=min(dis[u][0],dis[v][0]+edge[i].w);
        dis[u][1]=max(dis[u][1],dis[v][1]+edge[i].w);
        cnt[u]+=cnt[v];
    }
}
int main()
{
    int n=read_int(),root=1,u,v;
    _for(i,1,n){
        u=read_int(),v=read_int();
        Insert(u,v);Insert(v,u);
    }
    dfs_1(root,0,0);
    dfs_2(root,root);
    int q=read_int();
    while(q--){
        int k=read_int();
        _for(i,0,k){
            node[i]=read_int();
            is_key[node[i]]=true;
```

```
        }
        build(k);
        ans1=0,ans2=MAXN,ans3=0,tot=k;
        dp(root);
        printf("%lld %d %d\n",ans1,ans2,ans3);
        _for(i,0,k)
        is_key[node[i]]=false;
    }
    return 0;
}
```

习题二

洛谷p3233

题意

给定一棵 n 个节点的无根树，q 次询问。

每次询问给定 k_i 个关键点。树上每个节点属于离它最近的关键点(如果有多个距离最近点则选择编号最小的)。

每次询问要求输出 k_i 个数，代表属于每个给定关键点的节点数。

题解

强制以 1 为根，转化为有根树。先考虑暴力 dp 的做法。

在原树上，先一次 dfs 找到每个节点子树方向上距离最小的关键节点，同时记录最小节点的距离和编号。

然后再一次 dfs 更新每个节点祖先方向的距离最小的关键节点，即可得到答案。

暴力 dp 对每次询问时间复杂度 $O(n)$，接下来考虑虚树优化 dp。

对虚树上的点，直接使用上述暴力 dp 即可。关键在于如何计算不在虚树上的点对关键节点答案的贡献。

对于不在虚树上的点，要么位于虚树上的某个节点在原树上的儿子节点的子树中且该子树中没有关键节点，记为第一类点。

要么位于虚树上的某两个节点在原树上的路径上的某个节点的非路径方向的子树中，记为第二类点。

先考虑第一类点的贡献，发现第一类点可以全部计入它的第一个属于虚树的祖先节点的贡献中。

对于第二类点的贡献。取虚树上两相邻点在原树上的路径，可以根据到哪个关键节点近将路径分成两段。

每段路径上的节点及其非路径方向的子树对距离该节点近的关键节点做贡献，路径分界点可通过倍增查找。

具体实现过程与上述讨论存在少量差异，详细见代码。

```
const int MAXN=3e5+5,MAXM=20;
struct Edge{
    int to,w,next;
}edge[MAXN<<1];
int head[MAXN],edge_cnt;
void Insert(int u,int v,int w=1){
    edge[++edge_cnt]=Edge{v,w,head[u]};
    head[u]=edge_cnt;
}
int d[MAXN],anc[MAXN][MAXM],lg2[MAXN],sz[MAXN],dfn[MAXN],dfs_t;
void dfs(int u,int fa){
    anc[u][0]=fa,d[u]=d[fa]+1,sz[u]=1,dfn[u]=++dfs_t;
    for(int i=1;(1<<i)<d[u];i++)
    anc[u][i]=anc[anc[u][i-1]][i-1];
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(v==fa)
        continue;
        dfs(v,u);
        sz[u]+=sz[v];
    }
}
void init(int root){
    lg2[0]=-1;
    _for(i,1,MAXN)
    lg2[i]=lg2[i>>1]+1;
    d[root]=1;
    dfs(root,0);
}
int LCA(int p,int q){
    if(d[p]<d[q])
    swap(p,q);
    while(d[p]>d[q])p=anc[p][lg2[d[p]-d[q]]];
    if(p==q)
    return p;
    for(int i=lg2[d[p]-1];i>=0;i--){
        if(anc[p][i]!=anc[q][i])
        p=anc[p][i],q=anc[q][i];
    }
    return anc[p][0];
}
struct cmp{
    bool operator () (const int a,const int b)const{
        return dfn[a]<dfn[b];
    }
};
int Stack[MAXN],top,node[MAXN],temp[MAXN];
void build(int k){
```

```
    edge_cnt=0;
    sort(node,node+k,cmp());
    Stack[top=1]=1;head[1]=0;
    _for(i,0,k){
        if(node[i]==1)continue;
        int p=LCA(Stack[top],node[i]);
        if(p!=Stack[top]){
while(dfn[p]<dfn[Stack[top-1]])Insert(Stack[top-1],Stack[top],d[Stack[top]]
-d[Stack[top-1]]),top--;
            if(p!=Stack[top-1])
            head[p]=0,Insert(p,Stack[top],d[Stack[top]]-d[p]),Stack[top]=p;
            else
            Insert(Stack[top-1],Stack[top],d[Stack[top]]-
d[Stack[top-1]]),top--;
        }
        head[node[i]]=0,Stack[++top]=node[i];
    }
    while(top>1)Insert(Stack[top-1],Stack[top],d[Stack[top]]-
d[Stack[top-1]]),top--;
}
bool is_key[MAXN];
int min_dis[MAXN],min_node[MAXN],ans[MAXN];
void dfs_2(int u){
    if(is_key[u])min_dis[u]=0,min_node[u]=u;
    else min_dis[u]=MAXN+1,min_node[u]=0;
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        dfs_2(v);
        if(min_dis[u]>min_dis[v]+edge[i].w)
        min_dis[u]=min_dis[v]+edge[i].w,min_node[u]=min_node[v];
        else
if((min_dis[u]==min_dis[v]+edge[i].w)&&min_node[u]>min_node[v])
        min_node[u]=min_node[v];
    }
}
void cal(int p,int q){
    int son=q,cut=q;
    while(d[son]>d[p]+1)son=anc[son][lg2[d[son]-d[p]-1]];
    ans[min_node[p]]-=sz[son];
    for(int i=lg2[d[q]-d[p]];i>=0;i--){
        if(d[anc[cut][i]]<d[p])continue;
        int dis1=d[anc[cut][i]]-d[p]+min_dis[p],dis2=d[q]-
d[anc[cut][i]]+min_dis[q];
        if(dis1>dis2||(dis1==dis2&&min_node[p]>min_node[q]))
        cut=anc[cut][i];
    }
    ans[min_node[q]]+=sz[cut]-sz[q];ans[min_node[p]]+=sz[son]-sz[cut];
}
void dfs_3(int u){
    ans[min_node[u]]+=sz[u];
    for(int i=head[u];i;i=edge[i].next){
```

```
        int v=edge[i].to;
        if(min_dis[v]>min_dis[u]+edge[i].w)
        min_dis[v]=min_dis[u]+edge[i].w,min_node[v]=min_node[u];
        else
if((min_dis[v]==min_dis[u]+edge[i].w)&&min_node[v]>min_node[u])
        min_node[v]=min_node[u];
        cal(u,v);
        dfs_3(v);
    }
}
int main()
{
    int n=read_int(),root=1,u,v;
    _for(i,1,n){
        u=read_int(),v=read_int();
        Insert(u,v);Insert(v,u);
    }
    init(root);
    int q=read_int();
    while(q--){
        int k=read_int();
        _for(i,0,k){
            node[i]=temp[i]=read_int();
            is_key[node[i]]=true;ans[node[i]]=0;
        }
        build(k);
        dfs_2(root);
        dfs_3(root);
        _for(i,0,k){
            space(ans[temp[i]]);
            is_key[node[i]]=false;
        }
        puts("");
    }
    return 0;
}
```

习题三

牛客暑期多校(第一场) B 题

题意

定义一棵无限节点的树，任意大于 1 的节点 p 与节点 $\frac p{\text{mindiv}(p)}$ 连一条边，其中 $\text{mindiv}(p)$ 代表 p 的最小素因子。

多次询问，每次给定 $w_1,w_2\cdots w_m$求 $\min_u\sum_{i=1}^m w_i\text{dis}(u,i!)$

题解

将这棵树视为以 1 为根的有根树，记 $i!(1\le i\le m)$ 为关键节点。

显然如果 u 为最佳位置，那么 u 的子树中至少要有一个关键节点，否则令 $v=\text{fa}(u)$ 则 $\sum_{i=1}^m w_i\text{dis}(v,i!)\lt \sum_{i=1}^m w_i\text{dis}(u,i!)$。

于是可以将无限大的树转化为叶子节点只能是关键节点的有限大树。

假设虚树已经建立，考虑换根 dp 即可得到最佳答案。

接下来考虑如何建立虚树。

首先发现根节点到叶子节点的路径代表的素因子是不增的。

具体的，将一个数分解质因数并将质因子从小到大排列，得到 $2^{\alpha_1}3^{\alpha_2}5^{\alpha_3}\cdots p_k^{\alpha_k}$ 则根节点到该节点的路径为

$$\begin{matrix} \underbrace{p_k,p_k\cdots p_k} & \cdots & \underbrace{5,5\cdots 5} & \underbrace{3,3\cdots 3} & \underbrace{2,2\cdots 2} \\ \alpha_k & \cdots & \alpha_3 & \alpha_2 & \alpha_1\end{matrix}$$

由此不难得出，如果每次优先选择代表素因子小的路径遍历，则 $1!,2!,3!\cdots m!$ 的 dfs 序递增。

接下来考虑如何求相邻两数的 LCA 的深度，记 i 的最大素因子为 p。

根据式子 $i!=(i-1)!\ast i$ 不难发现根节点到 $(i-1)!$ 的路径与到 $i!$ 的路径在最开始的不包含 p 的部分完全相同。

而接下来的路径分叉点产生的原因为 $i!$ 比 $(i-1)!$ 拥有更多的 p 因子。据此不难求出 LCA 的深度，用树状数组维护即可。

所有 LCA 的深度可以提前预处理，总时间复杂度 $O(\max(m\log^2 m)+\sum m)$。

```
const int MAXP=1e5+5;
int prime[MAXP],mindiv[MAXP],cnt;
void Prime(){
    mindiv[1]=1;
    _for(i,2,MAXP){
        if(!mindiv[i])prime[cnt++]=i,mindiv[i]=i;
        for(int j=0;j<cnt&&i*prime[j]<MAXP;j++){
            mindiv[i*prime[j]]=prime[j];
            if(i%prime[j]==0) break;
        }
    }
}
#define lowbit(x) (x)&(-x)
int c[MAXP];
void add(int pos){
    while(pos<MAXP){
        c[pos]++;
```

```
        pos+=lowbit(pos);
    }
}
int query(int pos){
    int ans=0;
    while(pos){
        ans+=c[pos];
        pos-=lowbit(pos);
    }
    return ans;
}
int d[MAXP],lca_d[MAXP];
void get_dep(){
    int pos;
    d[1]=0;
    _for(i,2,MAXP){
        d[i]=d[i-1];
        for(pos=i;pos!=mindiv[pos];pos/=mindiv[pos])d[i]++;d[i]++;
        lca_d[i]=query(MAXP-1)-query(pos-1);
        for(pos=i;pos!=1;pos/=mindiv[pos])add(mindiv[pos]);
    }
}
const int MAXN=2e5+5;
struct Edge{
    int to,w,next;
}edge[MAXN<<1];
int head[MAXN],edge_cnt;
void Insert(int u,int v,int w){
    edge[++edge_cnt]=Edge{v,w,head[u]};
    head[u]=edge_cnt;
}
int Stack[MAXN],node_d[MAXN],top,tot;
void build(int k){
    edge_cnt=0;Stack[top=1]=1;head[1]=0;node_d[1]=0;tot=k;
    _rep(i,2,k){
        node_d[i]=d[i];
        if(lca_d[i]!=node_d[Stack[top]]){
            while(lca_d[i]<node_d[Stack[top-1]])
            Insert(Stack[top-1],Stack[top],node_d[Stack[top]]-
node_d[Stack[top-1]]),top--;
            if(lca_d[i]!=node_d[Stack[top-1]]){
                head[++tot]=0,node_d[tot]=lca_d[i];
                Insert(tot,Stack[top],node_d[Stack[top]]-node_d[tot]);
                Stack[top]=tot;
            }
            else
            Insert(Stack[top-1],Stack[top],node_d[Stack[top]]-
node_d[Stack[top-1]]),top--;
        }
```

```
        head[i]=0,Stack[++top]=i;
    }
    while(top>1)Insert(Stack[top-1],Stack[top],node_d[Stack[top]]-
node_d[Stack[top-1]]),top--;
}
int n,w[MAXN],s[MAXN];
LL ans[MAXN],re;
void dfs(int u){
    if(u<=n)s[u]=w[u],ans[1]+=w[u]*node_d[u];
    else s[u]=0;
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        dfs(v);
        s[u]+=s[v];
    }
}
void dp(int u){
    re=min(ans[u],re);
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(s[1]-s[v]<s[v]){
            ans[v]=ans[u]+(s[1]-s[v]*2)*edge[i].w;
            dp(v);
        }
    }
}
int main()
{
    int root=1;
    Prime();get_dep();
    while(~scanf("%d",&n)){
        _rep(i,1,n)
        w[i]=read_int();
        build(n);
        ans[1]=0,re=1e18;
        dfs(root);
        dp(root);
        enter(re);
    }
    return 0;
}
```

习题四

洛谷p4242

题意

给定一棵树，每个点有一种颜色。定义 $T(u,v)$ 表示点 u 到点 v 路径的颜色段数。接下来 q 个操作。

1. 将路径 $u\to v$ 的所有结点颜色修改为 c
2. 给定 m 个点 $a_1,a_2\cdots a_m$，对每个 i，询问 $\sum_{j=1}^m T(a_i,a_j)$

题解

建立虚树，同时定义 $u\to v$ 的权值为 $u\to v$ 的颜色段数减去最开始 u 颜色的一端。

树剖维护边修改和边询问，注意合并时需要分别维护左路径和右路径。最后点分治统计答案。时间复杂度 $O((\sum m)\log^2 n)$。

```
const int MAXN=1e5+5;
namespace T{
    struct Edge{
        int to,next;
    }edge[MAXN<<1];
    int head[MAXN],edge_cnt;
    void Insert(int u,int v){
        edge[++edge_cnt]=Edge{v,head[u]};
        head[u]=edge_cnt;
        edge[++edge_cnt]=Edge{u,head[v]};
        head[v]=edge_cnt;
    }
    struct Node{
        int lc,rc,sum;
        Node(int lc=0,int rc=0,int sum=0):lc(lc),rc(rc),sum(sum){}
        Node operator + (const Node &b){
            if(!sum)return b;
            if(!b.sum)return *this;
            return Node(lc,b.rc,sum+b.sum-(rc==b.lc));
        }
    }s[MAXN<<2];
    int a[MAXN],lef[MAXN<<2],rig[MAXN<<2],lazy[MAXN<<2];
    void push_up(int k){s[k]=s[k<<1]+s[k<<1|1];}
    void build(int k,int L,int R){
        lef[k]=L,rig[k]=R;
        int M=L+R>>1;
        if(L==R){
            s[k]=Node(a[M],a[M],1);
            return;
        }
        build(k<<1,L,M);
        build(k<<1|1,M+1,R);
        push_up(k);
    }
    void push_tag(int k,int c){
```

```
        s[k].lc=s[k].rc=c,s[k].sum=1;
        lazy[k]=c;
    }
    void push_down(int k){
        if(lazy[k]){
            push_tag(k<<1,lazy[k]);
            push_tag(k<<1|1,lazy[k]);
            lazy[k]=0;
        }
    }
    void update(int k,int L,int R,int v){
        if(L<=lef[k]&&rig[k]<=R){
            push_tag(k,v);
            return;
        }
        push_down(k);
        int mid=lef[k]+rig[k]>>1;
        if(mid>=L)
        update(k<<1,L,R,v);
        if(mid<R)
        update(k<<1|1,L,R,v);
        push_up(k);
    }
    Node query(int k,int L,int R){
        if(L<=lef[k]&&rig[k]<=R)
        return s[k];
        push_down(k);
        int mid=lef[k]+rig[k]>>1;
        if(mid<L)return query(k<<1|1,L,R);
        else if(mid>=R)return query(k<<1,L,R);
        else
        return query(k<<1,L,R)+query(k<<1|1,L,R);
    }
    int c[MAXN],d[MAXN],sz[MAXN],f[MAXN],dfn[MAXN],dfs_t;
    int h_son[MAXN],mson[MAXN],p[MAXN];
    void dfs_1(int u,int fa,int depth){
        sz[u]=1;f[u]=fa;d[u]=depth;mson[u]=0;
        for(int i=head[u];i;i=edge[i].next){
            int v=edge[i].to;
            if(v==fa)continue;
            dfs_1(v,u,depth+1);
            sz[u]+=sz[v];
            if(sz[v]>mson[u]){
                h_son[u]=v;
                mson[u]=sz[v];
            }
        }
    }
    void dfs_2(int u,int top){
        dfn[u]=++dfs_t;p[u]=top;a[dfs_t]=c[u];
        if(mson[u])
```

```
        dfs_2(h_son[u],top);
        for(int i=head[u];i;i=edge[i].next){
            int v=edge[i].to;
            if(v==f[u]||v==h_son[u])continue;
            dfs_2(v,v);
        }
    }
    int query_path(int u,int v){
        Node ansu,ansv;
        while(p[u]!=p[v]){
            if(d[p[u]]<d[p[v]])
            swap(u,v),swap(ansu,ansv);
            ansu=query(1,dfn[p[u]],dfn[u])+ansu;
            u=f[p[u]];
        }
        if(d[u]>d[v])swap(u,v),swap(ansu,ansv);
        swap(ansu.lc,ansu.rc);
        return (ansu+query(1,dfn[u],dfn[v])+ansv).sum-1;
    }
    void update_path(int u,int v,int w){
        while(p[u]!=p[v]){
            if(d[p[u]]<d[p[v]])
            swap(u,v);
            update(1,dfn[p[u]],dfn[u],w);
            u=f[p[u]];
        }
        if(d[u]>d[v])
        swap(u,v);
        update(1,dfn[u],dfn[v],w);
    }
    int LCA(int u,int v){
        while(p[u]!=p[v]){
            if(d[p[u]]<d[p[v]])
            swap(u,v);
            u=f[p[u]];
        }
        if(d[u]>d[v])
        swap(u,v);
        return u;
    }
}
struct Edge{
    int to,next,w;
}edge[MAXN<<1];
int head[MAXN],edge_cnt;
void Insert(int u,int v,int w){
    edge[++edge_cnt]=Edge{v,head[u],w};
    head[u]=edge_cnt;
    edge[++edge_cnt]=Edge{u,head[v],w};
```

```
    head[v]=edge_cnt;
}
struct cmp{
    bool operator () (const int a,const int b)const{
        return T::dfn[a]<T::dfn[b];
    }
};
int Stack[MAXN],top,temp[MAXN],node[MAXN];
void build(int k){
    edge_cnt=0;
    sort(node,node+k,cmp());
    Stack[top=1]=1;head[1]=0;
    _for(i,0,k){
        if(node[i]==1)continue;
        int p=T::LCA(Stack[top],node[i]);
        if(p!=Stack[top]){
            while(T::dfn[p]<T::dfn[Stack[top-1]])
Insert(Stack[top-1],Stack[top],T::query_path(Stack[top],Stack[top-1])),top-
-;
            if(p!=Stack[top-1])
head[p]=0,Insert(p,Stack[top],T::query_path(Stack[top],p)),Stack[top]=p;
            else
Insert(Stack[top-1],Stack[top],T::query_path(Stack[top],Stack[top-1])),top-
-;
        }
        head[node[i]]=0,Stack[++top]=node[i];
    }
while(top>1)Insert(Stack[top-1],Stack[top],T::query_path(Stack[top],Stack[t
op-1])),top--;
}
bool is_key[MAXN],vis[MAXN];
LL ans[MAXN],slen[MAXN],s1;
int sz[MAXN],mson[MAXN],tot_sz,root,root_sz,s2;
void find_root(int u,int fa){
    sz[u]=1;mson[u]=0;
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(vis[v]||v==fa)
        continue;
        find_root(v,u);
        sz[u]+=sz[v];
        mson[u]=max(mson[u],sz[v]);
    }
    mson[u]=max(mson[u],tot_sz-sz[u]);
    if(mson[u]<root_sz){
        root=u;
        root_sz=mson[u];
    }
}
void dfs_1(int u,int fa,int len){
    if(is_key[u])
```

```
    sz[u]=1,slen[u]=len;
    else
    sz[u]=slen[u]=0;
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(vis[v]||v==fa)
        continue;
        dfs_1(v,u,len+edge[i].w);
        sz[u]+=sz[v];
        slen[u]+=slen[v];
    }
}
void dfs_2(int u,int fa,int len){
    if(is_key[u])ans[u]+=s1+1LL*len*s2;
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(vis[v]||v==fa)
        continue;
        dfs_2(v,u,len+edge[i].w);
    }
}
void query(int u){
    dfs_1(u,0,1);
    ans[u]+=slen[u];
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(vis[v])
        continue;
        s1=slen[u]-slen[v];
        s2=sz[u]-sz[v];
        dfs_2(v,u,edge[i].w);
    }
}
void solve(int u){
    int cur_sz=tot_sz;
    vis[u]=true;query(u);
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(vis[v])
        continue;
        tot_sz=sz[v]>sz[u]?cur_sz-sz[u]:sz[v];root_sz=MAXN;
        find_root(v,u);
        solve(root);
    }
}
void dfs_3(int u,int fa){
    tot_sz++;
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
```

```
        if(v==fa)continue;
        dfs_3(v,u);
    }
}
void dfs_4(int u,int fa){
    ans[u]=0,vis[u]=is_key[u]=false;
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(v==fa)continue;
        dfs_4(v,u);
    }
}
int main()
{
    int n=read_int(),q=read_int(),rt=1;
    _rep(i,1,n)T::c[i]=read_int();
    _for(i,1,n){
        int u=read_int(),v=read_int();
        T::Insert(u,v);
    }
    T::dfs_1(rt,0,0);T::dfs_2(rt,1);
    T::build(1,1,n);
    while(q--){
        int opt=read_int();
        if(opt==1){
            int u=read_int(),v=read_int();
            T::update_path(u,v,read_int());
        }
        else{
            int k=read_int();
            _for(i,0,k){
                temp[i]=node[i]=read_int();
                is_key[node[i]]=true;
            }
            build(k);
            tot_sz=0;root_sz=MAXN;
            dfs_3(rt,0);
            find_root(rt,0);
            solve(root);
            _for(i,0,k)
            space(ans[temp[i]]);
            putchar('\n');
            dfs_4(rt,0);
        }
    }
    return 0;
}
```