// 树链剖分

int n,q;

int depth[100048],sz[100048],fa[100048];

int son[100048];int top[100048];

vector<int> v[100048];

int weight[100048],id[100048],ind=0;

int Pos[100048];

char type[10];

struct node

{

int left,right;

LL sum;int maxn;

}tree[300048];

//第一遍dfs：求出基本信息，size,son,fa,depth

void dfs(int cur,int father)

{

int i,max\_sz=-1;

sz[cur]=1;fa[cur]=father;son[cur]=0;

for (i=0;i<v[cur].size();i++)

if (v[cur][i]!=father)

{

depth[v[cur][i]]=depth[cur]+1;

dfs(v[cur][i],cur);

sz[cur]+=sz[v[cur][i]];

if (sz[v[cur][i]]>max\_sz)

{

son[cur]=v[cur][i];

max\_sz=sz[v[cur][i]];

}

}

}

//第二遍dfs:根据重边安排线段树

//tp存的是当前点的top(重链顶端顶点编号)

void dfs2(int cur,int tp)

{

int i;

top[cur]=tp;id[cur]=++ind;Pos[ind]=cur;

//优先搜索重儿子，使得重边在线段树中能连续存储

if (son[cur]) dfs2(son[cur],tp);

//搜索轻儿子

for (i=0;i<v[cur].size();i++)

if (v[cur][i]!=fa[cur] && v[cur][i]!=son[cur])

dfs2(v[cur][i],v[cur][i]);

}

void build(int cur,int left,int right)

{

tree[cur].left=left;tree[cur].right=right;

if (left!=right)

{

int mid=(left+right)>>1;

build(cur<<1,left,mid);

build(cur<<1|1,mid+1,right);

tree[cur].sum=tree[cur<<1].sum+tree[cur<<1|1].sum;

tree[cur].maxn=max(tree[cur<<1].maxn,tree[cur<<1|1].maxn);

}

else

tree[cur].maxn=tree[cur].sum=weight[Pos[left]];

}

void update(int cur,int pos,int nl)

{

if (tree[cur].left==tree[cur].right)

{

tree[cur].maxn=tree[cur].sum=nl;

return;

}

int mid=(tree[cur].left+tree[cur].right)>>1;

if (pos<=mid) update(cur<<1,pos,nl); else update(cur<<1|1,pos,nl);

tree[cur].maxn=max(tree[cur<<1].maxn,tree[cur<<1|1].maxn);

tree[cur].sum=tree[cur<<1].sum+tree[cur<<1|1].sum;

}

int query\_max(int cur,int left,int right)

{

if (left<=tree[cur].left && tree[cur].right<=right) return tree[cur].maxn;

int mid=(tree[cur].left+tree[cur].right)>>1;

int res=-INF;

if (left<=mid) res=max(res,query\_max(cur<<1,left,right));

if (mid+1<=right) res=max(res,query\_max(cur<<1|1,left,right));

return res;

}

LL query\_sum(int cur,int left,int right)

{

if (left<=tree[cur].left && tree[cur].right<=right) return tree[cur].sum;

LL res=0;

int mid=(tree[cur].left+tree[cur].right)>>1;

if (left<=mid) res+=query\_sum(cur<<1,left,right);

if (mid+1<=right) res+=query\_sum(cur<<1|1,left,right);

return res;

}

//求u-v的sum

LL calc\_sum(int u,int v)

{

LL res=0;

int tp1=top[u],tp2=top[v];

while (tp1!=tp2)

{

//注意这个地方容易写错：要比较top的高低，而不是u,v的高低

if (depth[tp1]<depth[tp2])

{

swap(u,v);

swap(tp1,tp2);

}

res+=query\_sum(1,id[tp1],id[u]);

u=fa[tp1];tp1=top[u];

}

if (depth[u]<depth[v]) swap(u,v);

res+=query\_sum(1,id[v],id[u]);

return res;

}

//求u-v的max

int calc\_max(int u,int v)

{

int res=-INF;

int tp1=top[u],tp2=top[v];

while (tp1!=tp2)

{

if (depth[tp1]<depth[tp2])

{

swap(u,v);

swap(tp1,tp2);

}

res=max(res,query\_max(1,id[tp1],id[u]));

u=fa[tp1];tp1=top[u];

}

if (depth[u]<depth[v]) swap(u,v);

res=max(res,query\_max(1,id[v],id[u]));

return res;

}

int main ()

{

int i,x,y,len;

n=getint();

for (i=1;i<=n-1;i++)

{

x=getint();y=getint();

v[x].pb(y);v[y].pb(x);

}

for (i=1;i<=n;i++) weight[i]=getint();

depth[1]=1;dfs(1,0);dfs2(1,1);

build(1,1,ind);

q=getint();

for (i=1;i<=q;i++)

{

scanf("%s",type+1);

if (type[2]=='H') //CHANGE 更新

{

x=getint();len=getint();

update(1,id[x],len);

}

if (type[2]=='S') //QSUM 求u-v的sum

{

x=getint();y=getint();

printf("%lld\n",calc\_sum(x,y));

}

if (type[2]=='M') //QMAX 求u-v的max

{

x=getint();y=getint();

printf("%d\n",calc\_max(x,y));

}

}

return 0;

}