```cpp
#include<bits/stdc++.h>
using namespace std;
const int MAXN=10002, LOGN=14, INF=2000000000, root=1;
struct dt{ int u,v,w; }edge[MAXN];
vector<int>adj[MAXN],tree[MAXN];
int a[MAXN],depth[MAXN],parent[MAXN],subSize[MAXN],sChild[MAXN];
int chainNo,head[MAXN],chainHead[MAXN],chainPos[MAXN],chainInd[MAXN],chainSize[MAXN];
void update_tree(int chid, int nd, int l, int r,int x,int val){
    if(l==x&&r==x){ tree[chid][nd] = val; return; }
    int lf = (nd<<1), rg = lf+1, m = (l+r)>>1;
    if(x<=m) update_tree(chid,lf,l,m,x,val);
    else update_tree(chid,rg,m+1,r,x,val);
    tree[chid][nd] = max(tree[chid][lf],tree[chid][rg]);
}
int query_tree(int chid, int nd, int l, int r,int x,int y){
    if(l>y||r<x)return -INF;
    if(l>=x&&r<=y)return tree[chid][nd];
    int lf = (nd<<1), rg = lf+1, m = (l+r)>>1;
    int p1 = query_tree(chid,lf,l,m,x,y);
    int p2 = query_tree(chid,rg,m+1,r,x,y);
    return max(p1,p2);
}
int query_up(int u,int lca){
    if(u==lca)return 0;
    int ans = -INF;
    while(depth[u]>depth[lca]){
        int l = (depth[head[u]]>depth[lca]) ? chainPos[head[u]] : chainPos[lca]+1;
        int r = chainPos[u];
        int sz = chainSize[chainInd[u]];
        int ret = query_tree(chainInd[u],1,1,sz,l,r);
        if(ret>ans)ans=ret;
        u = parent[head[u]];
    }
    return ans;
}
int getLCA(int u, int v){
    while(chainInd[u]!=chainInd[v]){
        if(depth[head[u]]>depth[head[v]]) u = parent[head[u]];
        else v = parent[head[v]];
    }
    if(depth[u]<depth[v])return u;
    else return v;
}
int query(int u,int v){
    int lca = getLCA(u,v);
    int ret1 = query_up(u, lca);
    int ret2 = query_up(v, lca);
    int ans = ret1 > ret2 ? ret1 : ret2;
    return ans;
}
void build_chain(int _head){
    chainNo++;
    chainHead[chainNo] = _head;
    int u = _head;
    vector<int>vv;
    while(u!=-1){
        vv.push_back(u);
        chainInd[u] = chainNo;
        chainPos[u] = (int)vv.size();
        head[u] = _head;
        u = sChild[u];
    }
    chainSize[chainNo] = (int)vv.size();
    tree[chainNo].resize(4*chainSize[chainNo], 0);
    for(int i=0; i<(int)vv.size(); i++){
        update_tree(chainNo,1,1,chainSize[chainNo],chainPos[vv[i]],a[vv[i]]);
    }
}
void HLD_DFS(int u,int prev){
    for(int i=0; i<adj[u].size(); i++){
        int v = adj[u][i];
        if(v!=prev){
            HLD_DFS(v,u);
        }
    }
```

```
    for(int i=0; i<adj[u].size(); i++){
        int v = adj[u][i];
        if(v != prev && v != sChild[u]){
            build_chain(v);
        }
    }
}
void HLD(){
    chainNo = 0;
    HLD_DFS(root,-1);
    build_chain(root);
}
void DFS(int u,int prev,int _depth){
    depth[u] = _depth; parent[u] = prev; subSize[u] = 1;
    int specialChild = -1, maxSize = 0;
    for(int i=0; i<adj[u].size(); i++){
        int v = adj[u][i];
        if(v!=prev){
            DFS(v,u,_depth+1);
            subSize[u] += subSize[v];
            if(subSize[v]>maxSize){
                specialChild = v;
                maxSize = subSize[v];
            }
        }
    }
    sChild[u] = specialChild;
}
void CLEAR(int n){
    for(int i=0; i<=n; i++) {
        adj[i].clear(); chainHead[i] = -1; head[i] = -1; sChild[i] = -1;
    }
}
int main(){
    int tt; scanf("%d",&tt);
    for(int ks=1; ks<=tt; ks++){
        int n; scanf("%d",&n);
        CLEAR(n);
        for(int i=1; i<n; i++){
            int u,v,w; scanf("%d%d%d",&u,&v,&w);
            adj[u].push_back(v);
            adj[v].push_back(u);
            edge[i].u=u; edge[i].v=v; edge[i].w=w;
        }

        DFS(root, -1, 0);

        for(int i=1; i<n; i++){
            if(depth[edge[i].u]>depth[edge[i].v])
                swap(edge[i].u,edge[i].v);
            a[edge[i].v] = edge[i].w;
        }

        HLD();

        while(1){
            char ss[10]; scanf("%s",&ss);
            if(ss[0]=='D')break;
            if(ss[0]=='Q'){
                int u,v; scanf("%d%d",&u,&v);
                int ans = query(u, v);
                printf("%d\n",ans);
            }
            else{
                int idx,weight; scanf("%d%d",&idx,&weight);
                int v = edge[idx].v;
                int sz = chainSize[chainInd[v]];
                update_tree(chainInd[v],1,1,sz,chainPos[v],weight);
            }
        }
    }
    return 0;
}
```