```cpp
#include<bits/stdc++.h>
using namespace std;
const int MAXN=10002, LOGN=14, INF=2000000000, root=1;
vector<int>adj[MAXN],cost[MAXN],indexx[MAXN];
int depth[MAXN],parent[MAXN],subSize[MAXN],otherEnd[MAXN],pa[MAXN][LOGN];
int chainNo,chainHead[MAXN],chainPos[MAXN],chainInd[MAXN],chainSize[MAXN];
int ptr,baseArray[MAXN],posInBase[MAXN], tree[4*MAXN];
void make_tree(int nd, int l, int r){
    if(l==r){ tree[nd] = baseArray[l]; return; }
    int lf = (nd<<1), rg = lf+1, m = (l+r)>>1;
    make_tree(lf,l,m);
    make_tree(rg,m+1,r);
    tree[nd] = max(tree[lf],tree[rg]);
}
void update_tree(int nd, int l, int r,int x,int val){
    if(l==x&&r==x){ tree[nd] = val; return; }
    int lf = (nd<<1), rg = lf+1, m = (l+r)>>1;
    if(x<=m) update_tree(lf,l,m,x,val);
    else update_tree(rg,m+1,r,x,val);
    tree[nd] = max(tree[lf],tree[rg]);
}
int query_tree(int nd, int l, int r,int x,int y){
    if(l>y||r<x)return -INF;
    if(l>=x&&r<=y)return tree[nd];
    int lf = (nd<<1), rg = lf+1, m = (l+r)>>1;
    int p1 = query_tree(lf,l,m,x,y);
    int p2 = query_tree(rg,m+1,r,x,y);
    return max(p1,p2);
}
int query_up(int u,int v){
    if(u==v)return 0;
    int uchain = chainInd[u], vchain = chainInd[v], ans = -INF;
    while(1){
        if(uchain==vchain){
            int ret = query_tree(1,1,ptr,posInBase[v]+1,posInBase[u]);
            if(ret>ans)ans=ret;
            break;
        }
        int ret = query_tree(1,1,ptr,posInBase[chainHead[uchain]],posInBase[u]);
        if(ret>ans)ans=ret;
        u = chainHead[uchain];
        u = parent[u];
        uchain = chainInd[u];
    }
    return ans;
}
int LCA(int u, int v){
    if(depth[u]<depth[v])swap(u,v);
    for(int i=LOGN-1; i>=0; i--){
        if(pa[u][i] != -1 && depth[u]-(1<<i) >= depth[v]){
            u = pa[u][i];
        }
    }
    if(u==v)return u;
    for(int i=LOGN-1; i>=0; i--){
        if(pa[u][i] != -1 && pa[u][i] != pa[v][i]){
            u = pa[u][i]; v = pa[v][i];
        }
    }
    return parent[u];
}
int query(int u,int v){
    int lca  = LCA(u,v);
    int ret1 = query_up(u, lca);
    int ret2 = query_up(v, lca);
    int ans  = ret1 > ret2 ? ret1 : ret2;
    return ans;
}
void change(int idx,int weight){
    int u = otherEnd[idx];
    update_tree(1, 1, ptr, posInBase[u], weight);
}
void preprocess_lca(int n){
    for(int u=1; u<=n; u++)pa[u][0]=parent[u];
    for(int i=1; i<LOGN; i++){
        for(int u=1; u<=n; u++){
            if(pa[u][i-1]!=-1){
                pa[u][i] = pa[pa[u][i-1]][i-1];
            }
        }
    }
}
```

```cpp
void HLD(int u,int prev,int weight){
    if(chainHead[chainNo]==-1) chainHead[chainNo]=u;
    chainSize[chainNo]++;
    chainInd[u] = chainNo; chainPos[u] = chainSize[chainNo];
    posInBase[u] = ++ptr; baseArray[ptr] = weight;
    int specialChild = -1, maxSize = 0, nweight;
    for(int i=0; i<adj[u].size(); i++){
        int v = adj[u][i];
        if(v!=prev){
            if(subSize[v]>maxSize){
                specialChild = v;
                maxSize = subSize[v];
                nweight = cost[u][i];
            }
        }
    }
    if(specialChild != -1){
        HLD(specialChild,u,nweight);
    }
    for(int i=0; i<adj[u].size(); i++){
        int v = adj[u][i];
        if(v != prev){
            if(v != specialChild){
                ++chainNo;
                HLD(v,u,cost[u][i]);
            }
        }
    }
}
void DFS(int u,int prev,int _depth=0){
    depth[u] = _depth; parent[u] = prev; subSize[u] = 1;
    for(int i=0; i<adj[u].size(); i++){
        int v = adj[u][i];
        int _index = indexx[u][i];
        if(v!=prev){
            otherEnd[_index] = v;
            DFS(v,u,_depth+1);
            subSize[u] += subSize[v];
        }
    }
}
void CLEAR(int n){
    for(int i=0; i<=n; i++) {
        adj[i].clear(); cost[i].clear(); indexx[i].clear(); chainHead[i]=-1;
        for(int j=0; j<LOGN; j++) pa[i][j] = -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); cost[u].push_back(w); indexx[u].push_back(i);
            adj[v].push_back(u); cost[v].push_back(w); indexx[v].push_back(i);
        }
        chainNo = 0, ptr = 0;
        DFS(root, -1);
        HLD(root, -1, -INF);
        make_tree(1, 1, ptr);
        preprocess_lca(n);
        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,wgt; scanf("%d%d",&idx,&wgt);
                change(idx, wgt);
            }
        }
    }
    return 0;
}
```