1348 - Aladdin and the Return Journey

SUBMIT	PDF (English)	Statistics	Forum
Time Limit: 2 second(s)		Memory Limit: 32 MB	

Finally the Great Magical Lamp was in Aladdin's hand. Now he wanted to return home. But he didn't want to take any help from the Genie because he thought that it might be another adventure for him. All he remembered was the paths he had taken to reach there. But since he took the lamp, all the genies in the cave became angry and they were planning to attack. As Aladdin was not afraid, he wondered how many genies were there. He summoned the Genie from the lamp and asked this.

Now you are given a similar problem. For simplicity assume that, you are given a tree (a connected graph with no cycles) with **n**nodes, nodes represent places, edges represent roads. In each node, initially there are an arbitrary number of genies. But the numbers of genies change in time. So, you are given a tree, the number of genies in each node and several queries of two types. They are:

1) **0 i j**, it means that you have to find the total number of genies in the **nodes** that occur in path from node **i** to **j** **($0 \le i, j < n$)**.
2) **1 i v,** it means that number of genies in node **i** is changed to **v** **($0 \le i < n, 0 \le v \le 1000$)**.

Input

Input starts with an integer **T** **(≤ 10)**, denoting the number of test cases.

Each case starts with a blank line. Next line contains an integer **n** **($2 \le n \le 30000$)**. The next line contains **n** space separated integers between **0** and **1000**, denoting the number of genies in the nodes respectively. Then there are **n-1** lines each containing two integers: **u v** **($0 \le u, v < n, u \ne v$)** meaning that there is an edge from node **u** and **v**. Assume that the edges form a valid tree. Next line contains an integer **q** **($1 \le q \le 10^5$)** followed by **q** lines each containing a query as described above.

Output

For each case, print the case number in a single line. Then for each query **0 i j**, print the total number of genies in the nodes that occur in path **i** to **j**.

Sample Input	Output for Sample Input
1 4 10 20 30 40 0 1 1 2 1 3 3 0 2 3 1 1 100 0 2 3	Case 1: 90 170

Note

Dataset is huge, use faster I/O methods.

```cpp
#include<bits/stdc++.h>
using namespace std;
const int MAXN=30002, LOGN=16, INF=2000000000, root=0;
vector<int>adj[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];
int a[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] = 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] = tree[lf]+tree[rg];
}
int query_tree(int nd, int l, int r,int x,int y){
    if(l>y||r<x)return 0;
    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 p1+p2;
}
int query_up(int u,int v){
    if(u==v)return 0;
    int uchain = chainInd[u], vchain = chainInd[v], ans = 0;
    while(1){
        if(uchain==vchain){
            int ret = query_tree(1,1,ptr,posInBase[v]+1,posInBase[u]);
            ans += ret;
            break;
        }
        int ret = query_tree(1,1,ptr,posInBase[chainHead[uchain]],posInBase[u]);
        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){
    if(u==v)return a[u];
    if(depth[u]<depth[v])swap(u,v);
    int lca = LCA(u,v);
```

```
67      int ret1=0, ret2=0;
68      if(u!=lca)ret1 = query_up(u, lca);
69      if(v!=lca)ret2 = query_up(v, lca);
70      int ans = ret1+ret2+a[lca];
71      return ans;
72  }
73  void change(int u,int weight){
74      a[u] = weight;
75      baseArray[posInBase[u]]=weight;
76      update_tree(1, 1, ptr, posInBase[u], weight);
77  }
78  void preprocess_lca(int n){
79      for(int u=0; u<n; u++)pa[u][0]=parent[u];
80      for(int i=1; i<LOGN; i++){
81          for(int u=0; u<n; u++){
82              if(pa[u][i-1]!=-1){
83                  pa[u][i] = pa[pa[u][i-1]][i-1];
84              }
85          }
86      }
87  }
88  void HLD(int u,int prev){
89      if(chainHead[chainNo]==-1) chainHead[chainNo]=u;
90      chainSize[chainNo]++;
91      chainInd[u] = chainNo;
92      chainPos[u] = chainSize[chainNo];
93      posInBase[u] = ++ptr;
94      baseArray[ptr] = a[u];
95
96      int specialChild = -1, maxSize = 0;
97
98      for(int i=0; i<adj[u].size(); i++){
99          int v = adj[u][i];
100         if(v!=prev){
101             if(subSize[v]>maxSize){
102                 specialChild = v;
103                 maxSize = subSize[v];
104             }
105         }
106     }
107
108     if(specialChild != -1){
109         HLD(specialChild,u);
110     }
111
112     for(int i=0; i<adj[u].size(); i++){
113         int v = adj[u][i];
114         if(v != prev){
115             if(v != specialChild){
116                 ++chainNo;
117                 HLD(v,u);
118             }
119         }
120     }
121 }
122 void DFS(int u,int prev,int _depth=0){
123     depth[u] = _depth;
124     parent[u] = prev;
125     subSize[u] = 1;
126     for(int i=0; i<adj[u].size(); i++){
127         int v = adj[u][i];
128         if(v!=prev){
129             DFS(v,u,_depth+1);
130             subSize[u] += subSize[v];
131         }
132     }
133 }
```

```
134 void CLEAR(int n){
135     for(int i=0; i<=n; i++) {
136         adj[i].clear();
137         chainHead[i] = -1;
138         for(int j=0; j<LOGN; j++) pa[i][j] = -1;
139     }
140 }
141 int main(){
142     int tt; scanf("%d",&tt);
143     for(int ks=1; ks<=tt; ks++){
144         printf("Case %d:\n",ks);
145         int n; scanf("%d",&n);
146         for(int i=0; i<n; i++)scanf("%d",&a[i]);
147         CLEAR(n);
148         for(int i=1; i<n; i++){
149             int u,v; scanf("%d%d",&u,&v);
150             adj[u].push_back(v);
151             adj[v].push_back(u);
152         }
153 
154         chainNo = 0, ptr = 0;
155 
156         DFS(root, -1);
157         HLD(root, -1);
158         make_tree(1, 1, ptr);
159         preprocess_lca(n);
160 
161         int q; scanf("%d",&q);
162         while(q--){
163             int choice; scanf("%d",&choice);
164             if(choice==0){
165                 int u,v; scanf("%d%d",&u,&v);
166                 int ans = query(u, v);
167                 printf("%d\n",ans);
168             }
169             else{
170                 int u,wgt; scanf("%d%d",&u,&wgt);
171                 change(u, wgt);
172             }
173         }
174     }
175     return 0;
176 }
```

```cpp
#include<bits/stdc++.h>
using namespace std;
const int MAXN=30002, LOGN=16, INF=2000000000, root=0;
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] = 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 0;
    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 p1+p2;
}
int query_up(int u,int lca){
    if(u==lca)return 0;
    int ans = 0;
    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);
        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){
    if(u==v)return a[u];
    if(depth[u]<depth[v])swap(u,v);
    int lca = getLCA(u,v);
    int ret1=0, ret2=0;
    if(u!=lca) ret1 = query_up(u, lca);
    if(v!=lca) ret2 = query_up(v, lca);
    int ans = ret1+ret2+a[lca];
    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]]);
    }
}
```

```
73  void HLD_DFS(int u,int prev){
74      for(int i=0; i<adj[u].size(); i++){
75          int v = adj[u][i];
76          if(v!=prev){
77              HLD_DFS(v,u);
78          }
79      }
80      for(int i=0; i<adj[u].size(); i++){
81          int v = adj[u][i];
82          if(v != prev && v != sChild[u]){
83              build_chain(v);
84          }
85      }
86  }
87  void HLD(){
88      chainNo = 0;
89      HLD_DFS(root,-1);
90      build_chain(root);
91  }
92  void DFS(int u,int prev,int _depth){
93      depth[u] = _depth; parent[u] = prev; subSize[u] = 1;
94      int specialChild = -1, maxSize = 0;
95      for(int i=0; i<adj[u].size(); i++){
96          int v = adj[u][i];
97          if(v!=prev){
98              DFS(v,u,_depth+1);
99              subSize[u] += subSize[v];
100             if(subSize[v]>maxSize){
101                 specialChild = v;
102                 maxSize = subSize[v];
103             }
104         }
105     }
106     sChild[u] = specialChild;
107 }
108 void CLEAR(int n){
109     for(int i=0; i<=n; i++) {
110         adj[i].clear(); chainHead[i] = -1; head[i] = -1; sChild[i] = -1;
111     }
112 }
113 int main(){
114     int tt; scanf("%d",&tt);
115     for(int ks=1; ks<=tt; ks++){
116         printf("Case %d:\n",ks);
117         int n; scanf("%d",&n);
118         for(int i=0; i<n; i++)scanf("%d",&a[i]);
119         CLEAR(n);
120         for(int i=1; i<n; i++){
121             int u,v; scanf("%d%d",&u,&v);
122             adj[u].push_back(v); adj[v].push_back(u);
123         }
124
125         DFS(root, -1, 0);
126         HLD();
127
128         int q; scanf("%d",&q);
129         while(q--){
130             int choice; scanf("%d",&choice);
131             if(choice==0){
132                 int u,v; scanf("%d%d",&u,&v);
133                 int ans = query(u, v);
134                 printf("%d\n",ans);
135             }
136             else{
137                 int u,weight; scanf("%d%d",&u,&weight);
138                 a[u] = weight;
139                 int sz = chainSize[chainInd[u]];
140                 update_tree(chainInd[u],1,1,sz,chainPos[u],weight);
141             }
142         }
143     }
144     return 0;
145 }
```