Hello World ! -- 图乱杂

/\*

树链剖分模板题

题意：一棵树上有n个节点，编号分别为1到n，每个节点都有一个权值w。我们将以下面的形式来要求你对这棵树完成一些操作： I. CHANGE u t : 把结点u的权值改为t II. QMAX u v: 询问从点u到点v的路径上的节点的最大权值 III. QSUM u v: 询问从点u到点v的路径上的节点的权值和 注意：从点u到点v的路径上的节点包括u和v本身

\*/

void find\_heavy\_edge(int u, int father, int dep)

{

depth[u] = dep;

fa[u] = father;

size[u] = 1;

int Max\_size = 0;

for(int k = head[u]; k != -1; k = edge[k].next) {

int v = edge[k].v;

if(v == father) continue;

find\_heavy\_edge(v, u, dep+1);

size[u] += size[v];

if(size[v] > Max\_size) Max\_size = size[v], hson[u] = v;

}

}

void connect\_heavy\_chain(int u, int ancester)

{

tid[u] = ++Index;

top[u] = ancester;

w[Index] = weight[u];

if(hson[u] != -1) connect\_heavy\_chain(hson[u], ancester);

for(int k = head[u]; k != -1; k = edge[k].next) {

int v = edge[k].v;

if(v == hson[u] || v == fa[u]) continue;

connect\_heavy\_chain(v, v);

}

}

int \_Query\_Max(int u, int l, int r)

{

if(l == f[u].l && f[u].r == r) return f[u].max;

if(r <= f[u].mid) return \_Query\_Max(L(u), l, r);

else if(f[u].mid < l) return \_Query\_Max(R(u), l, r);

else return max(\_Query\_Max(L(u), l, f[u].mid), \_Query\_Max(R(u), f[u].mid+1, r));

}

int \_Query\_Sum(int u, int l, int r)

{

if(l == f[u].l && f[u].r == r) return f[u].sum;

if(r <= f[u].mid) return \_Query\_Sum(L(u), l, r);

else if(f[u].mid < l) return \_Query\_Sum(R(u), l, r);

else return \_Query\_Sum(L(u), l, f[u].mid) + \_Query\_Sum(R(u), f[u].mid+1, r);

}

void push\_up(int u)

{

f[u].sum = f[L(u)].sum + f[R(u)].sum;

f[u].max = max(f[L(u)].max, f[R(u)].max);

}

void build(int u, int l, int r)

{

f[u].l = l, f[u].r = r, f[u].mid = MID(l,r);

if(l == r) {

f[u].max = f[u].sum = w[l];

return ;

}

build(L(u), l, f[u].mid), build(R(u), f[u].mid+1, r);

push\_up(u);

}

void Update(int u, int pos, int val)

{

if(f[u].l == f[u].r) {

f[u].max = f[u].sum = val;

return ;

}

if(pos <= f[u].mid) Update(L(u), pos, val);

else Update(R(u), pos, val);

push\_up(u);

}

int Query\_Max(int u, int v)

{

if(u == v) return \_Query\_Max(1, tid[u], tid[u]); //TODO

if(u == v) return 0;

int f1 = top[u], f2 = top[v], ans = -9999999;

while(f1 != f2) {

if(depth[f1] < depth[f2]) swap(f1, f2), swap(u, v);

ans = max(ans, \_Query\_Max(1, tid[f1], tid[u]));

u = fa[f1], f1 = top[u];

}

if(u == v) return max(ans, \_Query\_Max(1, tid[u], tid[u]));

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

return max(ans, \_Query\_Max(1, tid[u], tid[v]));

}

int Query\_Sum(int u, int v)

{

if(u == v) return \_Query\_Sum(1, tid[u], tid[u]);

int f1 = top[u], f2 = top[v], ans = 0;

while(f1 != f2) {

if(depth[f1] < depth[f2]) swap(f1, f2), swap(u, v);

ans += \_Query\_Sum(1, tid[f1], tid[u]);

u = fa[f1], f1 = top[u];

}

if(u == v) return ans + \_Query\_Sum(1, tid[u], tid[u]); //TODO

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

return ans + \_Query\_Sum(1, tid[u], tid[v]);

}

/\*

LCA 统计 好题 神题

题意：在一个无向树上给出若干条sample paths，最后输出每条边被多少条sample paths覆盖。

思路：暴力法是LCA直接暴搞，TLE。

然后想到像一维线段树那样标记法add[]搞，于是对于每条sample path(u,v)，标记add[u]++, add[v]++, add[ ancester(u,v) ] -= 2

但是后面就不知道怎么像一维线段树那样统计了。想了很久都没办法。然后看题解。

哇靠，太巧妙了！dfs回溯统计add，太妙了！！！赞！

//最后对整个图搜一遍，回溯的时候每个边加一下儿子几点add[]的值就可以了。

int dfs(int u, int fa)

{

for(int k = head[u]; k != -1; k = edge[k].next) {

int v = edge[k].v;

if(v == fa) continue;

times[ k/2+1 ] = dfs(v, u); //k/2+1是对应的边

add[u] += add[v];

}

return add[u];

}

\*/

#define ANC\_SIZE 30 //原本定为34，找了好久才发现错误，会导致LCA错误

void build(int u, int fa, int d)

{

depth[u] = d;

for(int k = head[u]; k != -1; k = edge[k].next) {

int v = edge[k].v;

if(v == fa) continue;

ancester[v][0] = u; //LCA的初始化ancester[x][0] = father[x]

up[v] = k^1;

build(v, u, d+1);

}

if( edge[ head[u] ].next == -1) leaf[u] = true;

}

int Find\_Anc(int u, int v)

{

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

int k = depth[u] - depth[v];

for(int i = 0; i < ANC\_SIZE; i++)

if(k & (1 << i)) u = ancester[u][i];

if(u == v) return v;

for(int i = ANC\_SIZE - 1; i >= 0; i--)

if(ancester[u][i] != ancester[v][i]) {

u = ancester[u][i], v = ancester[v][i];

}

return ancester[u][0];

}

void calc(int u, int v)

{

add[Find\_Anc(u, v)] -= 2;

add[u]++, add[v]++;

}

void Anc\_init()

{

for(int j = 1; j < ANC\_SIZE; j++)

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

ancester[i][j] = ancester[ ancester[i][j-1] ][j-1];

}

//void solve(int u) //这是我之前统计的办法，是错的。

//{

// int sum = add[u];

// while(u != 1) {

// times[ up[u]/2 + 1] += sum;

// u = edge[ up[u] ].v;

// sum += add[u];

// }

//}

int dfs(int u, int fa) //超妙的dfs！！！

{

for(int k = head[u]; k != -1; k = edge[k].next) {

int v = edge[k].v;

if(v == fa) continue;

times[ k/2+1 ] = dfs(v, u);

add[u] += add[v];

}

return add[u];

}

int main()

{

int u, v, query;

scanf("%d", &n);

init();

for(int i = 1; i < n; i++) {

scanf("%d%d", &u, &v);

add\_edge(u, v), add\_edge(v, u);

}

up[1] = -1;

build(1, 0, 0);

Anc\_init();

scanf("%d", &query);

while(query--) {

scanf("%d%d", &u, &v);

calc(u, v);

}

dfs(1, 0);

for(int i = 1; i < n; i++) printf("%d ", times[i]);

printf("\n");

return 0;

}

/\*

LCA 并查集 Tarjan离线 好题 难题 神题！！！

题意：

收获：并查集在压缩路径的时候可以更新一些数据，这点是我之前没见过的。 （详见Find()）

Tarjan离线搞最小公共祖先以递归回到祖先lca的时候才能有信息回答query(u,v)，之前都是遍历到v点的时候就能回答query(u,v)（假设u点已经遍历过了），这道题是要递归回到祖先z的时候才能回答。

思路：对于每个节点x，当前祖先为z

max(x)/min(x) x到z路径上的最大/小值

up(x) x->z路径上的dp值

down(x) z->x路径上的dp值

则ans( query(u,v) ) = max{ up(x), down(y), max(y)-min(x) }

方程不是重点，重点是如何实现.

代码解释：

//================================================================================

int Find(int x)

//把更新过程放在Find里面，边向上并查集找祖先，边更新x-lca的信息，把并查集用得太好了！！

//先把当前x的父亲y记录下来，然后father[x] = Find(Father[x]) 向上找y的祖先并更新节点y的属性。

//之后利用y点的信息更新x点信息。想象一下，超美妙。假设当前的祖先为z，则Find(father[x])过程中更新了区间(y, z)的信息，存在节点y里面，那接下来就用区间(y, z)的信息（存在节点y），和(x, y)的信息（存在节点x）来更新区间(x, z)的信息，并存到x节点。

{

if(x == father[x]) return x;

int y = father[x];

father[x] = Find(father[x]);

a[x].up = max(a[y].max - a[x].min, max(a[x].up, a[y].up));

a[x].down = max(a[x].max - a[y].min, max(a[x].down, a[y].down));

a[x].max = max(a[x].max, a[y].max);

a[x].min = min(a[x].min, a[y].min);

return father[x];

}

//==================================================================================

for(int Size = Quetion[u].size(), i = 0; i < Size; i++)

//以前的LCA都是对于query(u,v)，若vis[v] == true，则根据Find(v)就可以对query进行回答，但这道题不行。因为我们接下来要递归下去，要保持当前子树的祖先都是x，所以我们必须得在递归回来之后才能Union(x, fa)。那不Union，根据Find(v)找到当前v的祖先（当前树的根节点）又没有办法回答query(u,v)，那就不能像一般LCA那样在这里就回答咯~~

//怎么办呢？首先我们应该考虑，什么时候我们才能回答query(u,v)？设当前lca(u,v) = z，分析一下知道，要回答query(u,v)，就要知道(u->z)和(z->v)的信息。那自然我们就知道---当递归回到z的时候我们能回答query(u,v).

//所以这里我们把每个query(u,v)存到Q[z]里面，在LCA递归回到z的时候再输出。

{

int v = Quetion[u][i].first, id = Quetion[u][i].second;

if(!vis[v]) continue;

int fv = Find(v);

if(id >= 0) Q[fv].push\_back(u), Q[fv].push\_back(v), Q[fv].push\_back(id);

else Q[fv].push\_back(v), Q[fv].push\_back(u), Q[fv].push\_back(-id);

}

for(int k = head[u]; k != -1; k = edge[k].next) {

int v = edge[k].v;

if(!vis[v]) {

LCA(v);

Union(u, v); //Union的位置很重要！递归回来才能Union

}

}

for(int Size = Q[u].size(), i = 0; i < Size; i += 3) {

//这里就是对于每个z的回答. 这里u表示lca(x,y) = u

int x = Q[u][i], y = Q[u][i+1], id = Q[u][i+2];

Find(x), Find(y);

ans[ id ] = max( a[x].up, max(a[y].down, a[y].max - a[x].min) );

}

\*/

//poj 3728

int Find(int x)

{

if(x == father[x]) return x;

int y = father[x];

father[x] = Find(father[x]);

a[x].up = max(a[y].max - a[x].min, max(a[x].up, a[y].up));

a[x].down = max(a[x].max - a[y].min, max(a[x].down, a[y].down));

a[x].max = max(a[x].max, a[y].max);

a[x].min = min(a[x].min, a[y].min);

return father[x];

}

void Union(int u, int v)

{

father[v] = u;

}

void LCA(int u)

{

vis[u] = true;

for(int Size = Quetion[u].size(), i = 0; i < Size; i++) {

int v = Quetion[u][i].first, id = Quetion[u][i].second;

if(!vis[v]) continue;

int fv = Find(v);

if(id >= 0) Q[fv].push\_back(u), Q[fv].push\_back(v), Q[fv].push\_back(id);

else Q[fv].push\_back(v), Q[fv].push\_back(u), Q[fv].push\_back(-id);

}

for(int k = head[u]; k != -1; k = edge[k].next) {

int v = edge[k].v;

if(!vis[v]) {

LCA(v);

Union(u, v); //Union的位置很重要！

}

}

for(int Size = Q[u].size(), i = 0; i < Size; i += 3) {

int x = Q[u][i], y = Q[u][i+1], id = Q[u][i+2];

Find(x), Find(y);

ans[ id ] = max( a[x].up, max(a[y].down, a[y].max - a[x].min) );

}

}

int main()

{

int u, v, query;

while(scanf("%d", &n) != EOF) {

init();

for(int i = 1; i <= n; i++) {

scanf("%d", &a[i].min);

a[i].max = a[i].min;

a[i].up = a[i].down = 0;

}

for(int i = 1; i < n; i++) {

scanf("%d%d", &u, &v);

add\_edge(u, v), add\_edge(v, u);

}

scanf("%d", &query);

for(int i = 1; i <= query; i++) {

scanf("%d%d", &u, &v);

Quetion[u].push\_back( PII(v, i) ), Quetion[v].push\_back( PII(u, -i) );

}

LCA(1);

for(int i = 1; i <= query; i++) printf("%d\n", ans[i]);

}

return 0;

}

/\*

树转线性结构 LCA 树形结构转线性结构经典题

题意：给你一棵树，现在有两种操作，一个是修改u到v路径上点的权值(全部增加一个值)，另外一个是查询一个点的点权。

思路：考虑将树形结构转化为线性结构，记录l[x]和r[x]表示先序遍历时开始访问x的时间戳l[x]和结束访问x的时间戳r[x]（以1 为根）

对于修改(u,v)路径上的点权操作，我们将[ 1, l[u] ]和[ 1, l[v] ]区间上的值全都加上w，在[1, l[ LCA(u,v) ]]和[1, l[LCA(u,v)]-1]区间减去w，

然后查询点u则回答weight[u] + Query(l[u], l[u]) - Q(r[u], r[u])

摘抄：http://blog.csdn.net/lyhypacm/article/details/6734748

我们现在要做的是更新区间查找点，利用树状数组维护括号序列的增量(注意这里是增量，初始化全部为0)，我们这么考虑，对于每一个更新操作，假设更新的是u到v路径上的点权全部加w，那么对于u，我们做的是将1到l(u)位置上的元素全部加w，对于v我们做类似的处理。

你会发现，更新1到l(u)上的元素值的时候，对于一个结点x,如果x是u的祖先，那么x一定是左括号被增加了w，如果x不是其祖先，比如说u为4的时候，3不是4的祖先，那么x一定是左右括号都被增加了w。通过这个例子，大家应该都知道怎么做了吧，对于每次更新，我们将1到l(u)以及1到l(v)元素全部加w，然后将u和v的lca的左括号的位置找出来，假设为l，那么1到l-1的全部元素减去2w，因为我们要更新的是路径上的值，这时候我们会发现lca会被更新了两次，所以lca对应的左括号位置元素减w，这样我们就完成了一次更新操作。

对于查询操作，看了上面一段话，应该会知道怎么做了，一个点的点权增加量就是等于其左括号的增加量减去右括号的增加量，这样我们就可以利用一个树状数组配合在线LCA在O(NlogN)的时间内得到所有的查询的答案。

void predfs(int u, int father)

{

l[u] = ++Time;

for(int k = head[u]; k != -1; k = edge[k].next) {

int v = edge[k].v;

if(v == father) continue;

predfs(v, u);

}

r[u] = ++Time;

}

void preLCA(int u, int father, int dep)

{

depth[u] = dep+1;

for(int k = head[u]; k != -1; k = edge[k].next) {

int v = edge[k].v;

if(v == father) continue;

anc[v][0] = u;

for(int i = 1; i < ANC\_SIZE; i++)

anc[v][i] = anc[ anc[v][i-1] ][i-1];

preLCA(v, u, dep+1);

}

}

int LCA(int u, int v)

{

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

int k = depth[u] - depth[v];

for(int i = 0; i < ANC\_SIZE; i++) if(k & (1 << i)) {

u = anc[u][i];

}

if(u == v) return u;

for(int i = ANC\_SIZE - 1; i >= 0; i--)

if(anc[u][i] != anc[v][i])

u = anc[u][i], v = anc[v][i];

return anc[u][0];

}

void build(int u, int l, int r);

int Query(int u, int l, int r);

void Update(int u, int l, int r, int val);

int main()

{

int query, u, v, w;

char op[10];

while(scanf("%d%\*d%d", &n, &query) != EOF) {

init();

for(int i = 1; i <= n; i++) scanf("%d", &weight[i]);

for(int i = 1; i < n; i++) {

scanf("%d%d", &u, &v);

add\_edge(u, v), add\_edge(v, u);

}

predfs(1, 0);

preLCA(1, 0, 0);

build(1, 1, Time);

while(query--) {

scanf("%s", op);

if(op[0] == 'I' || op[0] == 'D') {

scanf("%d%d%d", &u, &v, &w);

if(op[0] == 'D') w = -w;

Update(1, 1, l[u], w), Update(1, 1, l[v], w);

int lca = LCA(u, v);

if(l[lca] > 1) Update(1, 1, l[lca]-1, -w);

Update(1, 1, l[lca], -w);

} else {

scanf("%d", &u);

printf("%d\n", weight[u] + Query(1, l[u], l[u]) - Query(1, r[u], r[u]));

}

}

}

return 0;

}

void build(int u, int l, int r)

{

f[u].l = l, f[u].r = r, f[u].lazy = f[u].num = 0;

if(l == r) return ;

int mid = MID(l, r);

build(L(u), l, mid), build(R(u), mid+1, r);

}

void put\_up(int u)

{

f[u].num = f[L(u)].num + f[R(u)].num;

}

void push\_down(int u)

{

int mid = MID(f[u].l, f[u].r);

Update(L(u), f[u].l, mid, f[u].lazy), Update(R(u), mid+1, f[u].r, f[u].lazy);

f[u].lazy = 0;

}

void Update(int u, int l, int r, int val)

{

if(l == f[u].l && f[u].r == r) {

f[u].lazy += val;

f[u].num += (r - l + 1 ) \* val;

return ;

}

if(f[u].lazy) push\_down(u);

int mid = MID(f[u].l, f[u].r);

if(r <= mid) Update(L(u), l, r, val);

else if(mid < l) Update(R(u), l, r, val);

else Update(L(u), l, mid, val), Update(R(u), mid+1, r, val);

put\_up(u);

}

int Query(int u, int l, int r)

{

if(l == f[u].l && r == f[u].r) {

return f[u].num;

}

if(f[u].lazy) push\_down(u);

int mid = MID(f[u].l, f[u].r);

if(r <= mid) return Query(L(u), l, r);

else if(mid < l) return Query(R(u), l,r);

else return Query(L(u), l, mid) + Query(R(u), mid+1, r);

}

/\*

树链剖分模板题

题意：一棵树上有n个节点，编号分别为1到n，每个节点都有一个权值w。我们将以下面的形式来要求你对这棵树完成一些操作： I. CHANGE u t : 把结点u的权值改为t II. QMAX u v: 询问从点u到点v的路径上的节点的最大权值 III. QSUM u v: 询问从点u到点v的路径上的节点的权值和 注意：从点u到点v的路径上的节点包括u和v本身

=\_=|| bzoj\_1036稍微改一下就1A了。

\*/

/\*

树链剖分模板题

题意：一棵树，两种操作，一是修改边i的权值，二是问(u,v)之间最大的边权

思路：树链剖分。我是把边权转为点权做的，对于边(u,v,w)，假设u是v的父节点，则weight[v] = w. 这样做需要注意的一点是，题目询问(u,v)的最大边权即询问(son[u], v)的最大点权，而不是(u,v)的最大点权。所以Query()的最后一句是 max(ans, \_Query(1, tid[son[u]], tid[v]));

\*/

void find\_heavy\_edge(int u, int father, int dep)

{

depth[u] = dep;

fa[u] = father;

size[u] = 1;

int Max = 0;

for(int k = head[u]; k != -1; k = edge[k].next) {

int v = edge[k].v;

if(v == father) continue;

find\_heavy\_edge(v, u, dep + 1);

weight[v] = edge[k].w; // 我想把边权化为点权

size[u] += size[v];

if(size[v] > Max) {

Max = size[v], son[u] = v;

}

}

}

void connect\_heavy\_chain(int u, int ancester)

{

tid[u] = ++Index;

F[Index] = u; //把剖分后的序列当作欧拉序列的话，那F[]就是对应的欧拉序列

top[u] = ancester; //top[u]表示u所在重链的头

if(son[u] != -1) connect\_heavy\_chain(son[u], ancester);

for(int k = head[u]; k != -1; k = edge[k].next) {

int v = edge[k].v;

if(v == fa[u] || v == son[u]) continue;

connect\_heavy\_chain(v, v);

}

}

int Query(int u, int v)

{

int f1 = top[u], f2 = top[v], ans = 0;

while(f1 != f2) {

if(depth[f1] < depth[f2]) swap(f1, f2), swap(u, v);

ans = max(ans, \_Query(1, tid[f1], tid[u]));

u = fa[f1], f1 = top[u];

}

if(u == v) return ans;

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

return max(ans, \_Query(1, tid[son[u]], tid[v]));

}

void push\_up(int u)

{

f[u].max = max(f[L(u)].max, f[R(u)].max);

}

void \_Update(int u, int pos, int val)

{

if(f[u].l == f[u].r) {

f[u].max = val;

return ;

}

int mid = MID(f[u].l, f[u].r);

if(pos <= mid) \_Update(L(u), pos, val);

else \_Update(R(u), pos, val);

push\_up(u);

}

int \_Query(int u, int l, int r)

{

if(l == f[u].l && f[u].r == r) return f[u].max;

int mid = MID(f[u].l, f[u].r);

if(r <= mid) return \_Query(L(u), l, r);

else if(mid < l) return \_Query(R(u), l, r);

else return max(\_Query(L(u), l, mid), \_Query(R(u), mid+1, r));

}

void Update(int e, int val)

{

int u = edge[(e-1)\*2].u, v = edge[(e-1)\*2].v;

if(fa[u] == v) swap(u, v);

\_Update(1, tid[v], val);

}

void build(int u, int l, int r)

{

f[u].l = l, f[u].r = r;

if(l == r) {

f[u].max = weight[ F[l] ];

return ;

}

int mid = MID(l, r);

build(L(u), l, mid), build(R(u), mid+1, r);

push\_up(u);

}