遥远的国度

一. 考察内容：

树 树上操作 树的换根 树链剖分 线段树

二. 题目分析：

[题目大意]

维护一棵树，要求实现三种操作：修改一条链上点的权值、换根、查询一个子树权值最小值。

[写题思路]

这道题有三个操作，其中如果没有换根操作，就是树链剖分模板题。

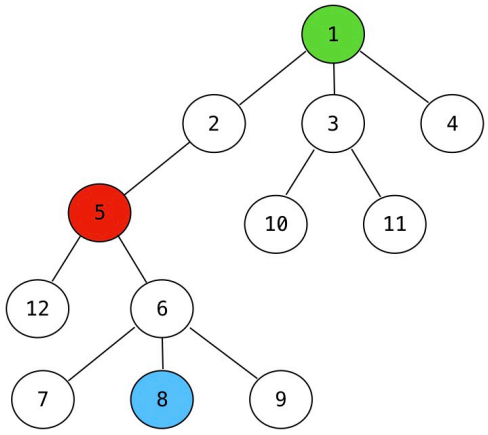
我们考虑如何实现换根操作。

首先我们明确，题目中的换根操作，不能真的O(n)换根，否则是会被卡掉的，我们考虑一种更快的实现方法。讨论对于操作1和操作3，换根会影响什么。很显然，操作一是要求对一条链进行修改，无论根在什么地方，修改的节点是不会变的，按照正常的方法用线段树维护树链剖分即可实现。

对于操作3，换根会显然会影响一些节点的子树元素，我们细分为三种情况讨论换根之后的影响：

S_1：在原图上，当前查询的节点为当前根的祖先

我们先来看一张图，绿色的节点是原来的根，蓝色的节点是当前的根，红色的节点是当前查询的节点，我们可以发现，5是8的祖先，这时，查询的子树范围是所有节点除去5通向8的整棵子树（即子树6），我们只需要用树剖找到5的子节点中通向8的节点，并将它剔除，再求最小值即可。



S_2：当前查询的节点就是根

这种情况实际查询的就是整个树的最小值，用线段树查询实现即可。

S_3：除S_1、S_2之外的所有情况

通过观察我们可以发现，这种情况的节点的子树元素是不会改变的，直接按照根为原根的方式求出子树最小值即可。

三. 代码实现：

```
#define _CRT_SECURE_NO_DEPRECATE
/************************
*创建时间：2018 08 30
*文件类型：源代码文件
*题目来源：COGS
*当前状态：已通过
*备忘录：树 树上操作 树的换根 树链剖分 线段树
*作者：HtBest
************************/
#include <stdio.h>
#include <iostream>
#include <string>
#include <stdlib.h>
#include <string.h>
#include <algorithm>
#include <queue>
// #include <sys/wait.h>
// #include <sys/types.h>
// #include <unistd.h>
using namespace std;
#define MAXN 100001
int n,m,root,head[MAXN],_edge,v[MAXN],
son[MAXN],fa[MAXN],bro[MAXN],
dfsx[MAXN],_dfsx,top[MAXN],num[MAXN],deep[MAXN],h_son[MAXN],
ndfsx[MAXN],cnt;
struct EDGE
{
    int a,b,next;
    EDGE(int a=0,int b=0,int next=0):a(a),b(b),next(next){}
}edge[2*MAXN];
struct NODE
{
    int lson,rson,v,lazy;
    void merge(NODE &a,NODE &b)
    {
        v=min(a.v,b.v);
    }
    void reset(){lson=rson=v=lazy=0;}
    void addl(int x)
    {
        v=lazy=x;
    }
}o[2*MAXN];
struct segmentTree
{
    int root;
    void downl(NODE &x)
    {
        if(x.lazy)
        {
            o[x.lson].addl(x.lazy);
            o[x.rson].addl(x.lazy);
        }
        x.lazy=0;
    }
    void build(int &x,int l,int r)
    {
        x=++cnt;
        o[x].reset();
        if(l==r)o[x].v=v[ndfsx[l]];
        else
        {
            int m=(l+r)>>1;
            build(o[x].lson,l,m);
            build(o[x].rson,m+1,r);
            o[x].merge(o[o[x].lson],o[o[x].rson]);
        }
    }
```

```
    void update(int x,int l,int r,int ul,int ur,int v)
    {
        if(l>=ul&&r<=ur)o[x].addl(v);
        else
        {
            int m=(l+r)>>1;
            downl(o[x]);
            if(ul<=m)update(o[x].lson,l,m,ul,ur,v);
            if(ur>m)update(o[x].rson,m+1,r,ul,ur,v);
            o[x].merge(o[o[x].lson],o[o[x].rson]);
        }
    }
    void query(int x,int l,int r,int ql,int qr,NODE &q)
    {
        if(l>=ql&&r<=qr)q=o[x];
        else
        {
            int m=(l+r)>>1;
            downl(o[x]);
            if(qr<=m)query(o[x].lson,l,m,ql,qr,q);
            else if(ql>m)query(o[x].rson,m+1,r,ql,qr,q);
            else
            {
                NODE a,b;
                query(o[x].lson,l,m,ql,qr,a);
                query(o[x].rson,m+1,r,ql,qr,b);
                q.merge(a,b);
            }
        }
    }
    void build(){cnt=0,build(root,1,n);}
    void update(int l,int r,int v){update(root,1,n,l,r,v);}
    int query(int l,int r)
    {
        if(l>r||l<=0)return 1e9;
        NODE ans;
        query(root,1,n,l,r,ans);
        return ans.v;
    }
}seg;
/* Variable explain:
n:节点数
m:操作数
head[i]:第i个节点的第一条出边
edge:邻接表
_edge:邻接表标记
v[i]:第i个点的权值
son[i]:第i个点的第一个儿子
fa[i]:第i个点的父亲
bro[i]:第i个点的下一个兄弟
dfsx[i]:节点i的dfs序
_dfsx:dfs序标记
ndfsx[i]:dfs序为i的节点序号
top[i]:第i个点的链顶
num[i]:子树i的节点个数
deep[i]:节点i的深度
h_son[i]:节点i的重儿子
cnt:线段树节点标记
o[i]:线段树的第i个节点
seg:线段树
*/
void adde(int a,int b)
{
    edge[++_edge]=EDGE(a,b,head[a]);
    head[a]=_edge;
}
void addt(int a,int b)
```

```
{
    bro[b]=son[a];
    son[a]=b;
    fa[b]=a;
}
void read()
{
    int ls1,ls2;
    scanf("%d%d",&n,&m);
    for(int i=1;i<n;++i)scanf("%d%d",&ls1,&ls2),adde(ls1,ls2),adde(ls2,ls1);
    for(int i=1;i<=n;++i)scanf("%d",&v[i]);
    scanf("%d",&root);
    return;
}
void dfs1(int a,int c)
{
    deep[a]=c;
    for(int i=head[a];i;i=edge[i].next)
    {
        int b=edge[i].b;
        if(fa[a]==b)continue;
        addt(a,b);
        dfs1(b,c+1);
        num[a]+=num[b];
        if(num[h_son[a]]<num[b])h_son[a]=b;
    }
    ++num[a];
}
void dfs2(int a,int t)
{
    dfsx[a]=++_dfsx;
    top[a]=t;
    if(h_son[a])dfs2(h_son[a],t);
    for(int i=son[a];i;i=bro[i])
        if(h_son[a]!=i)dfs2(i,i);
}
int lca1(int a,int b)
{
    while(1)
    {
        if(top[a]==top[b])break;
        if(deep[top[a]]>deep[top[b]])swap(a,b);
        b=fa[top[b]];
    }
    if(deep[a]>deep[b])swap(a,b);
    return a;
}
int lca2(int a,int b)
{
    while(1)
    {
        if(top[a]==top[b])break;//同链
        if(fa[top[b]]==a){b=top[b];break;}//异链
        b=fa[top[b]];
    }
    if(top[a]==top[b])
    {
        return min
        (
            seg.query(1,dfsx[h_son[a]]-1),
            seg.query(dfsx[h_son[a]]+num[h_son[a]],n)
        );
    }
    else
    {
        return min
        (
            seg.query(1,dfsx[b]-1),
            seg.query(dfsx[b]+num[b],n)
        );
    }
}
```

```
void lca(int a,int b,int v)
{
    while(1)
    {
        if(top[a]==top[b])break;
        if(deep[top[a]]>deep[top[b]])swap(a,b);
        seg.update(dfsx[top[b]],dfsx[b],v);
        b=fa[top[b]];
    }
    if(deep[a]>deep[b])swap(a,b);
    seg.update(dfsx[a],dfsx[b],v);
}
int main()
{
    // freopen("bbbbb.in","r",stdin);
    // freopen("bbbbb.out","w",stdout);
    read();
    dfs1(1,0);
    dfs2(1,1);
    for(int i=1;i<=n;++i)ndfsx[dfsx[i]]=i;
    seg.build();
    while(m--)
    {
        int op,ls1,ls2,ls3;
        scanf("%d",&op);
        switch(op)
        {
            case 1:
                scanf("%d",&ls1);
                root=ls1;
                break;
            case 2:
                scanf("%d%d%d",&ls1,&ls2,&ls3);
                lca(ls1,ls2,ls3);
                break;
            case 3:
                scanf("%d",&ls1);
                if(ls1==root)
                {
                    printf("%d\n",seg.query(1,n));
                }
                else if(lca1(ls1,root)==ls1)
                {
                    printf("%d\n",lca2(ls1,root));
                    // printf("%d\n",min(seg.query(1,dfsx[root]-1),seg.query(dfsx[root]
+num[root])))
                }
                else printf("%d\n",seg.query(dfsx[ls1],dfsx[ls1]+num[ls1]-1));
        }
    }
    return 0;
}
```

<题目跳转> <查看代码>