洛谷SP16549 QTREE6 - Query on a tree VI（LCT）

洛谷题目传送门

思路分析

题意就是要维护同色连通块大小。要用LCT维护子树大小就不说了，可以看看蒟蒻的LCT总结。
至于连通块如何维护，首先肯定可以想到一个很naive的做法：直接维护同色连通块，每次更改时暴力修改父边和子边。。。。。。
来个菊花图吧！（话说我真的好弱，前几天ZJOI的时候才知道对于某点度数很大的树/图有这样的称呼，真是很形象哈23333）
既然这条路行不通，那就换一种模型吧。
这是一种高级的维护染色连通块的较为通用的模型。
感觉蒟蒻对这种模型的理解与许多巨佬有不一样的地方，在这儿瞎扯扯吧
很多与树有关的题目，当边权不好处理时，有时候会转化为此边子节点的点权处理。
因为有根树中除了根，每个点都有唯一的父边。
在这一题里，道理是一样的，但转化方向却是反的，要把点化为边！
把每个点的父边赋予该点的颜色。我们需要两个LCT，每种对应一个颜色。一条边只有在对应颜色的LCT中才会被连上。
于是，原来同色点的连通块，就变成了剪开顶端节点后的边的连通块（解释一下，因为点的颜色给了父边，那么既然是顶端节点，那它的父边就不会在连通块中，也就是这个点与连通块不同色，于是该点的所有子树不能连起来，于是要剪掉）
然后就可以惊讶地发现，修改一个点的颜色之后，只要在原来颜色对应LCT中断掉父边，再在新颜色对应LCT中连接父边，就可以轻而易举地维护连通块啦。
再谈查询，上面提到了要剪开顶端节点（也就是连通块构成的树的树根），于是先findroot，再输出它的重子树的大小。
一个小细节，1节点是没有父亲的，不过为了模型的建立，要有父边，于是需要加一个虚点，让1的父亲指向它连边。

```
#include<cstdio>
#include<cstdlib>
#define R register int
#define I inline void
const int N=1000009,M=N<<1;
#define lc c[x][0]
#define rc c[x][1]
#define C col[u]
int fa[N],he[N],ne[M],to[M];
bool col[N];
struct LCT{
    int f[N],c[N][2],si[N],s[N],h[N];
    bool r[N];
    LCT(){for(R i=1;i<N;++i)s[i]=1;}//注意初始化
    inline bool nroot(R x){return c[f[x]][0]==x||c[f[x]][1]==x;}
    I pushup(R x){
        s[x]=s[lc]+s[rc]+si[x]+1;
    }
    I rotate(R x){
        R y=f[x],z=f[y],k=c[y][1]==x,w=c[x][!k];
        if(nroot(y))c[z][c[z][1]==y]=x;c[x][!k]=y;c[y][k]=w;
        f[w]=y;f[y]=x;f[x]=z;
        pushup(y);
    }
    I splay(R x){
        R y;
        while(nroot(x)){
            if(nroot(y=f[x]))rotate((c[f[y]][0]==y)^(c[y][0]==x)?x:y);
```

```
            rotate(x);
        }
        pushup(x);
    }
    I access(R x){
        for(R y=0;x;x=f[y=x]){
            splay(x);
            si[x]+=s[rc];
            si[x]-=s[rc=y];
        }
    }
    inline int findroot(R x){
        access(x);splay(x);
        while(lc)x=lc;
        splay(x);
        return x;
    }
    I link(R x){//只传一个参数，因为只会连父边，cut同理
        splay(x);//不用access(x)，因为x一定是连通块的根
        R y=f[x]=fa[x];
        access(y);splay(y);//与常规LCT不同，别忘加
        si[y]+=s[x];s[y]+=s[x];
    }
    I cut(R x){
        access(x);splay(x);
        lc=f[lc]=0;
        pushup(x);
    }
}lct[2];
void dfs(R x){
    for(R y,i=he[x];i;i=ne[i])
        if((y=to[i])!=fa[x])
            fa[y]=x,dfs(y),lct[0].link(y);
}
#define G ch=getchar()
#define in(z) G;\
    while(ch<'-')G;\
    z=ch&15;G;\
    while(ch>'-')z*=10,z+=ch&15,G
int main(){
    register char ch;
    R p=1,n,m,i,u,v,op;
    in(n);
    for(i=1;i<n;++i){
        in(u);in(v);
        to[++p]=v;ne[p]=he[u];he[u]=p;
        to[++p]=u;ne[p]=he[v];he[v]=p;
    }
    dfs(1);
    fa[1]=n+1;lct[0].link(1);//虚点
    in(m);
    while(m--){
        in(op);in(u);
        if(op)lct[C].cut(u),lct[C^=1].link(u);
        else{
            v=lct[C].findroot(u);
            printf("%d\n",lct[C].s[lct[C].c[v][1]]);
        }
    }
    return 0;
}
```

分类: 数据结构——链剖——LCT , OI——题解

标签: LCT