# 洛谷P3703 [SDOI2017]树点涂色（LCT，dfn序，线段树，倍增LCA）

洛谷题目传送门

## 闲话

这是所有LCT题目中的一个异类。
之所以认为是LCT题目，是因为本题思路的瓶颈就在于如何去维护同颜色的点的集合。
只不过做着做着，感觉后来的思路（dfn序，线段树，LCA）似乎要喧宾夺主了。。。（至少在代码上看是如此）

## 思路分析

一个一个操作来（瞎BB中，这种思路模式并不具有普遍性。。。。。。）

## 1操作

~~还好我没学树剖233333~~以至于~~（直接想到）~~只好用LCT来维护颜色。
题目透露出的神奇的性质——每一种颜色，无论在任何时刻，肯定是一条链，而且点的深度严格递增！
而且还特意指定根节点！1操作特意修改x到根节点的颜色！
想到了这里，就不难想到本题的关键模型——LCT中每个Splay辅助树维护同颜色点的集合
于是1操作==access。。。。。。

## 2操作

x到y路径？split？！
~~I'm too young too simple~~
LCT维护了集合，就只能维护集合了。随便再乱搞一下集合就被破坏了。
于是就要再外部维护了。
至于维护什么，现在其实还不能产生很好的思路。。。。。
但我们可以先想到一点：没有了LCT，还要资磁树中任意两点之间的询问?
常见的复杂度正确的方法~~（套路）~~能想到的就只有树上差分了吧（设F为状态，那么就形如F[x]+F[y]-F[lca]）
于是就可以只维护每个点到根节点路径上的颜色种数
转化一下，在LCT中就等价于每个点到根节点路径所要经过的轻边总数，这里又是一个关键点
查询F[x]+F[y]-2F[lca]+1（+1是因为lca所在的颜色被减了两次）
不会树剖，只好写倍增LCA
然后就可以转而思考如何维护这个状态了
首先初始状态就是每个点的深度，然后就接着考虑修改了。
因为状态只与轻边有关，所以在access时更改就可以啦
可以类比一下LCT维护子树信息和（可参考一下Blog的LCT总结）
access中有替换右儿子的操作，等于把原来一条边变轻，新的一条边变重
那么原来那条边所指的子树状态全部要+1（多了一个轻边），新连上的边所指的子树状态全部要-1
于是问题又出现了。。。
众所周知，LCT可以维护子树信息，但不可以修改子树信息
树剖很好维护就不提了，然后我又不会树剖TOT（我太弱了）
在这紧要关头，dfn序救了我。。。。。。
一个子树，所有的点的dfn序一定是连续的区间
所以维护线段树，表示dfn序的区间，修改的时候在对应的区间修改，查询就单点查

## 3操作

所有的思路难点，在操作2冗长的思路分析中都攻克了
这里就在线段树里维护状态最大值（树剖也一样），区间查询就好啦

至于写法，线段树里的区间加减法可以写懒标记，也可以实现永久化标记（YL巨佬做法，常数暴踩本蒟蒻，目前rank1）
只不过我试了一下，dfn序线段树写永久化标记因为某些无法描述的玄学问题变得更慢了。。。。。。

思路就这样，有些细节在代码里（Debug一晚上带来的惨痛的经验。。。。。。）
算上in，pup，pdn和main，此程序一共有15个函数。。。。。。

```
#include<cstdio>
#include<iostream>
using namespace std;
#define I inline
#define R register int
#define G ch=getchar()
#define in(z) G;\
    while(ch<'-')G;\
    z=ch&15;G;\
    while(ch>'-')z*=10,z+=ch&15,G
#define lc x<<1
#define rc x<<1|1
#define pup mx[x]=max(mx[lc],mx[rc])
#define pdn if(lz[x])upd(lc,l[x],m[x],lz[x]),upd(rc,m[x]+1,r[x],lz[x]),lz[x]
//都是线段树操作，本题的LCT内部没有维护信息
const int N=100009,M=N*20;
int l[M],m[M],r[M],mx[M],lz[M],f[N],c[N][2],st[N][20],o[N];
int p=1,he[N],ne[N<<1],to[N<<1],d[N],dfn[N],at[N],mr[N],now;
//at是dfn的反表示，mr表示每个点的子树在dfn序区间中的右端点（左端点是它自己）
I void dfs(R x,R fa){//建树预处理
    d[now=at[dfn[x]=++p]=x]=d[st[x][0]=f[x]=fa]+1;//一堆信息的预处理压进了一行
    for(R&i=o[x];(st[x][i+1]=st[st[x][i]][i]);++i);//倍增LCA预处理
    for(R i=he[x];i;i=ne[i])
        if(fa!=to[i])dfs(to[i],x);
    mr[x]=now;
}
I int lca(R x,R y){//求LCA
    if(d[x]<d[y])swap(x,y);
    for(R i=o[x];i>=0;--i)
        if(d[st[x][i]]>=d[y])x=st[x][i];//Debug中的错误1：>=写成了>
    if(x==y)return x;
    for(R i=o[x];i>=0;--i)
        if(st[x][i]!=st[y][i])x=st[x][i],y=st[y][i];
    return st[x][0];
}
I void build(R x,R s,R e){//建线段树
    l[x]=s;r[x]=e;m[x]=(s+e)>>1;
    if(s==e){mx[x]=d[at[s]];return;}//利用反表示找到初始状态
    build(lc,s,m[x]);build(rc,m[x]+1,e);pup;
}
I void upd(R x,R s,R e,R v){//区间修改
    if(l[x]==s&&r[x]==e){mx[x]+=v;lz[x]+=v;return;}//注意mx也要变
    pdn;
    if(e<=m[x])upd(lc,s,e,v);
    else if(s>m[x])upd(rc,s,e,v);
    else upd(lc,s,m[x],v),upd(rc,m[x]+1,e,v);
    pup;
}
I int get(R s){//单点查值，与区间查值分开了，为了减小常数
    R x=1;
    while(l[x]!=r[x]){
        pdn;x=(lc)+(s>m[x]);
    }
    return mx[x];
}
I int ask(R x,R s,R e){//区间查值
    if(l[x]==s&&r[x]==e)return mx[x];
    pdn;
```

```
    if(e<=m[x])return ask(lc,s,e);
    if(s>m[x])return ask(rc,s,e);
    return max(ask(lc,s,m[x]),ask(rc,m[x]+1,e));
}
I bool nrt(R x){//LCT部分
    return c[f[x]][0]==x||c[f[x]][1]==x;
}
I void rot(R x){
    R y=f[x],z=f[y],k=c[y][1]==x,w=c[x][!k];
    if(nrt(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;//Debug中的错误2：y写成了x
}
I void splay(R x){
    R y;
    while(nrt(x)){
        if(nrt(y=f[x]))rot((c[f[y]][0]==y)^(c[y][0]==x)?x:y);
        rot(x);
    }
}
I int frt(R x){//有别于传统意义下的findroot
    while(c[x][0])x=c[x][0];
    return x;
}
I void access(R x){
    for(R w,y=0;x;x=f[y=x]){
        splay(x);
        if(c[x][1])w=frt(c[x][1]),upd(1,dfn[w],dfn[mr[w]],1);
        if((c[x][1]=y))w=frt(y),upd(1,dfn[w],dfn[mr[w]],-1);
//Debug中的错误3：这里更新要找原子树的根（即深度最小的那个点）
//而不能把辅助树的根当原子树根直接upd(1,dfn[c[x][1]],dfn[mr[c[x][1]]],1)
    }
}
int main(){
    register char ch;
    R n,m,i,a,b,op,x,y;
    in(n);in(m);
    for(i=1;i<n;++i){
        in(a);in(b);
        to[++p]=b;ne[p]=he[a];he[a]=p;
        to[++p]=a;ne[p]=he[b];he[b]=p;
    }
    p=0;dfs(1,0);build(1,1,n);
    while(m--){
        in(op);in(x);
        if(op==1)access(x);
        else if(op==2){
            in(y);
            printf("%d\n",get(dfn[x])+get(dfn[y])-get(dfn[lca(x,y)])*2+1);
        }
        else printf("%d\n",ask(1,dfn[x],dfn[mr[x]]));
    }
    return 0;
}
```

分类: 数据结构——链剖——LCT , 数据结构——线段树 , 算法——倍增——倍增LCA , 图论——树——树的dfn序 , OI——题解

标签: LCT , 线段树 , 倍增LCA