LCT总结——概念篇+洛谷P3690[模板]Link Cut Tree(动态树)（LCT，Splay）

为了优化体验（~~其实是强迫症~~），蒟蒻把总结拆成了两篇，方便不同学习阶段的Dalao们切换。

LCT总结——应用篇戳这里

概念、性质简述

首先介绍一下**链剖分**的概念（感谢laofu的讲课）
链剖分，是指一类对树的边进行轻重划分的操作，这样做的目的是为了减少某些链上的修改、查询等操作的复杂度。
目前总共有三类：重链剖分，实链剖分和并不常见的长链剖分

重链剖分

实际上我们经常讲的树剖，就是重链剖分的常用称呼。
对于每个点，选择最大的子树，将这条连边划分为重边，而连向其他子树的边划分为轻边。
若干重边连接在一起构成重链，用树状数组或线段树等静态数据结构维护。
至于有怎样优秀的性质等等，不在本总结的讨论范畴了（其实是因为本蒟蒻连树剖都不会）

实链剖分

同样将某一个儿子的连边划分为实边，而连向其他子树的边划分为虚边。
区别在于虚实是可以动态变化的，因此要使用更高级、更灵活的Splay来维护每一条由若干实边连接而成的实链。
基于性质更加优秀的实链剖分，LCT(Link-Cut Tree)应运而生。
LCT维护的对象其实是一个森林。
在实链剖分的基础下，LCT资磁更多的操作

- 查询、修改链上的信息（最值，总和等）
- 随意指定原树的根（即换根）
- 动态连边、删边
- 合并两棵树、分离一棵树（~~跟上面不是一毛一样吗~~）
- 动态维护连通性
- 更多意想不到的操作（可以往下滑一滑）

想学Splay的话，推荐巨佬yyb的博客

LCT的主要性质如下：

1. 每一个Splay维护的是一条从上到下按在原树中深度严格递增的路径，且中序遍历Splay得到的每个点的深度序列严格递增。
   ~~是不是有点抽象哈~~
   比如有一棵树，根节点为1（深度1），有两个儿子$2,3$（深度2），那么Splay有3种构成方式：
   $\{1-2\},\{3\}$
   $\{1-3\},\{2\}$
   $\{1\},\{2\},\{3\}$（每个集合表示一个Splay）
   而不能把$1,2,3$同放在一个Splay中（存在深度相同的点）

2. 每个节点包含且仅包含于一个Splay中

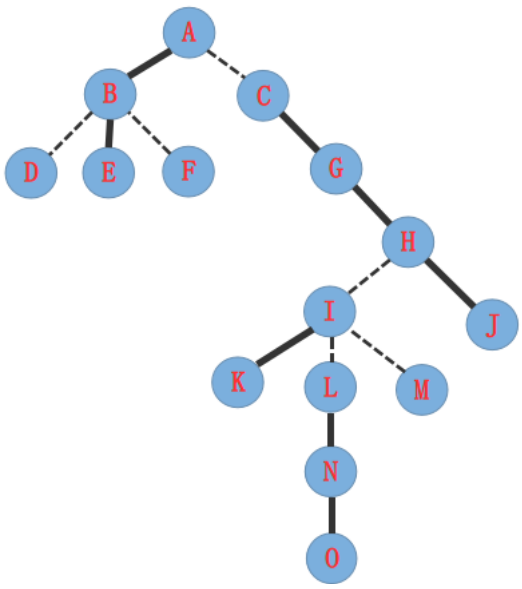
3. 边分为实边和虚边，实边包含在Splay中，而虚边总是由一棵Splay指向另一个节点（指向该Splay中中序遍历最靠前的点在原树中的父亲）。
   因为性质2，当某点在原树中有多个儿子时，只能向其中一个儿子拉一条实链（只认一个儿子），而其它儿子是不能在这个Splay中的。

那么为了保持树的形状，我们要让到其它儿子的边变为虚边，由对应儿子所属的Splay的根节点的父亲指向该点，而从该点并不能直接访问该儿子（认父不认子）。

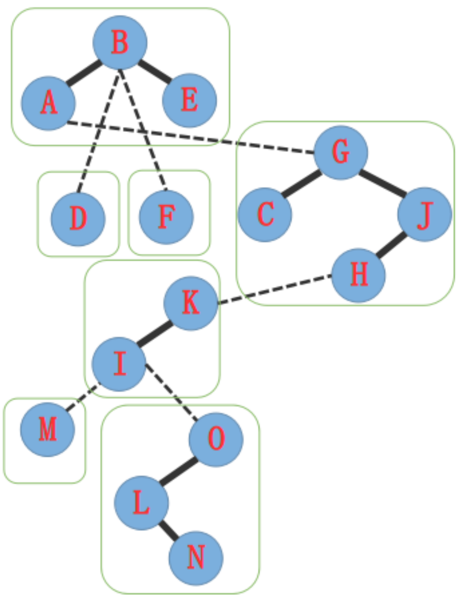
各种操作

$access(x)$

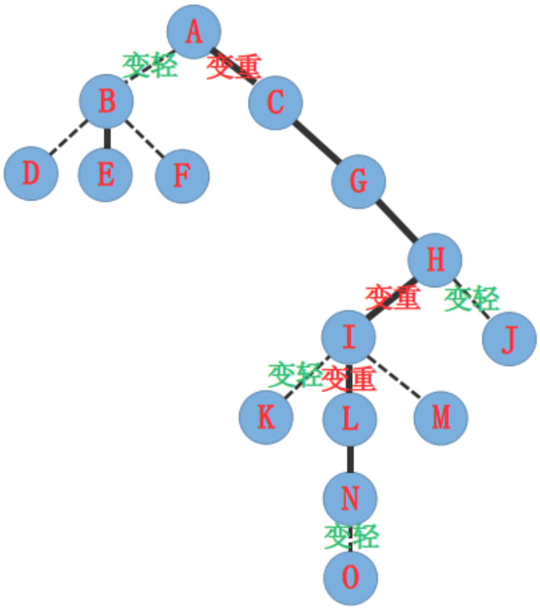
LCT核心操作，也是最难理解的操作。其它所有的操作都是在此基础上完成的。
因为性质3，我们不能总是保证两个点之间的路径是直接连通的（在一个Splay上）。
access即定义为打通根节点到指定节点的实链，使得一条中序遍历以根开始、以指定点结束的Splay出现。
~~蒟蒻深知没图的痛苦qwq~~
所以还是来几张图吧。
下面的图片参考YangZhe的论文
有一棵树，假设一开始实边和虚边是这样划分的（虚线为虚边）



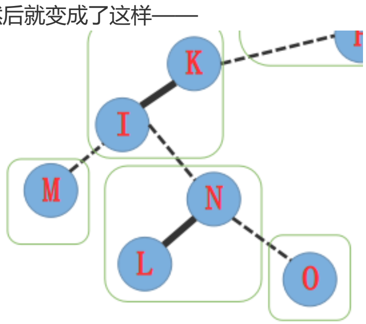
那么所构成的LCT可能会长这样（绿框中为一个Splay，可能不会长这样，但只要满足中序遍历按深度递增（性质1）就对结果无影响）



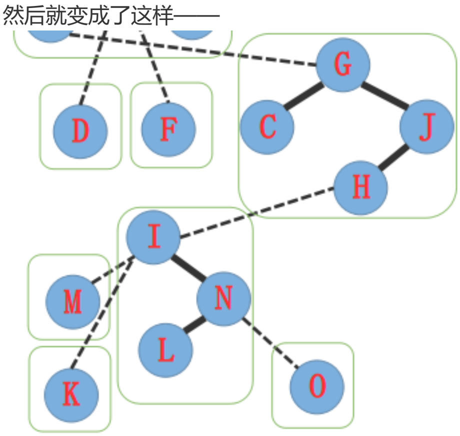
现在我们要$access(N)$，把$A-N$的路径拉起来变成一条Splay。
因为性质2，该路径上其它链都要给这条链让路，也就是把每个点到该路径以外的实边变虚。
所以我们希望虚实边重新划分成这样。



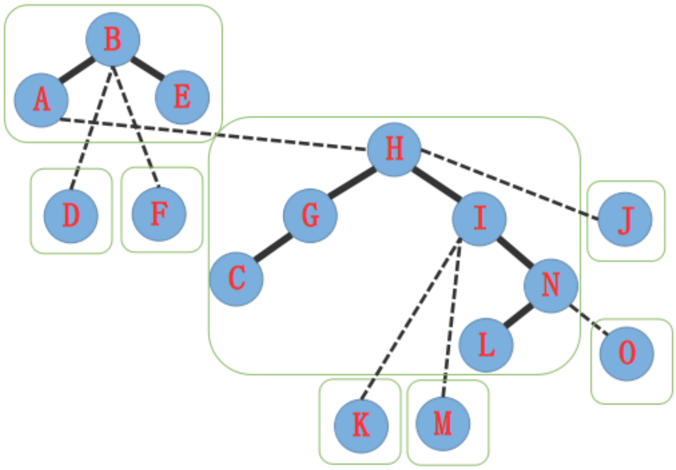
然后怎么实现呢?
我们要一步步往上拉。
首先把$splay(N)$，使之成为当前Splay中的根。
为了满足性质2，原来$N-O$的重边要变轻。
因为按深度O在N的下面，在Splay中O在N的右子树中，所以直接单方面将N的右儿子置为0（认父不认子）
然后就变成了这样——



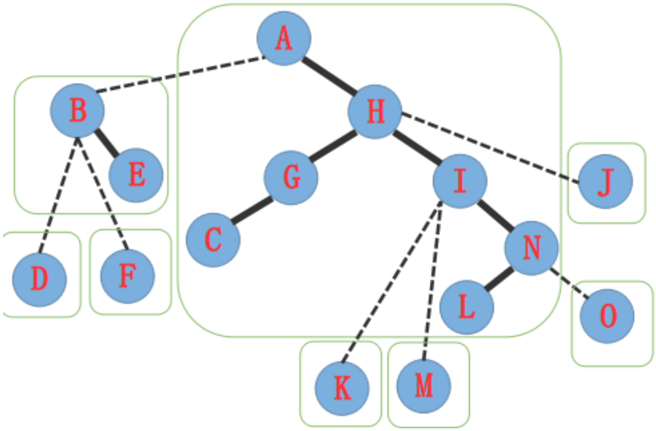
我们接着把N所属Splay的虚边指向的I（在原树上是L的父亲）也转到它所属Splay的根，$splay(I)$。
原来在I下方的重边$I-K$要变轻（同样是将右儿子去掉）。
这时候$I-L$就可以变重了。因为L肯定是在I下方的（刚才L所属Splay指向了I），所以I的右儿子置为N，满足性质1。
然后就变成了这样——



I指向H，接着$splay(H)$，H的右儿子置为I。



H指向A，接着$splay(A)$，A的右儿子置为H。



$A-N$的路径已经在一个Splay中了，大功告成！
代码其实很简单。。。。。。循环处理，只有四步——

1. 转到根；
2. 换儿子；
3. 更新信息；
4. 当前操作点切换为轻边所指的父亲，转1

```
inline void access(int x){
    for(int y=0;x;y=x,x=f[x])
        splay(x),c[x][1]=y,pushup(x);//儿子变了，需要及时上传信息
}
```

$makeroot(x)$

只是把根到某个节点的路径拉起来并不能满足我们的需要。更多时候，我们要获取指定两个节点之间的路径信息。
然而一定会出现路径不能满足按深度严格递增的要求的情况。根据性质1，这样的路径不能在一个Splay中。
Then what can we do?
$makeroot$定义为换根，让指定点成为原树的根。
这时候就利用到$access(x)$和Splay的翻转操作。
$access(x)$后x在Splay中一定是深度最大的点对吧。
$splay(x)$后，x在Splay中将没有右子树（性质1）。于是翻转整个Splay，使得所有点的深度都倒过来了，x没了左子树，反倒成了深度最小的点（根节点），达到了我们的目的。
代码

```
inline void pushr(int x){//Splay区间翻转操作
    swap(c[x][0],c[x][1]);
    r[x]^=1;//r为区间翻转懒标记数组
}
```

```
inline void makeroot(int x){
    access(x);splay(x);
    pushr(x);
}
```

关于pushdown和makeroot的一个相关的小问题详见下方update（关于pushdown的说明）

$findroot(x)$

找x所在原树的树根，主要用来判断两点之间的连通性（findroot(x)==findroot(y)表明x, y在同一棵树中）
代码：

```
inline int findroot(R x){
    access(x); splay(x);
    while(c[x][0])pushdown(x),x=c[x][0];
//如要获得正确的原树树根，一定pushdown！详见下方update（关于findroot中pushdown的
    splay(x);//保证复杂度
    return x;
}
```

同样利用性质1，不停找左儿子，因为其深度一定比当前点深度小。

$split(x, y)$

神奇的$makeroot$已经出现，我们终于可以访问指定的一条在原树中的链啦！
split(x,y)定义为拉出$x-y$的路径成为一个Splay（本蒟蒻以y作为该Splay的根）
代码

```
inline void split(int x,int y){
    makeroot(x);
    access(y);splay(y);
}
```

x成为了根，那么x到y的路径就可以用$access(y)$直接拉出来了，将y转到Splay根后，我们就可以直接通过访问y来获取该路径的有关信息

$link(x, y)$

连一条$x-y$的边（本蒟蒻使x的父亲指向y，连一条轻边）
代码

```
inline bool link(int x,int y){
    makeroot(x);
    if(findroot(y)==x)return 0;//两点已经在同一子树中，再连边不合法
    f[x]=y;
    return 1;
}
```

如果题目保证连边合法，代码就可以更简单

```
inline void link(int x,int y){
    makeroot(x);
    f[x]=y;
}
```

$cut(x, y)$

将$x-y$的边断开。
如果题目保证断边合法，倒是很方便。
使x为根后，y的父亲一定指向x，深度相差一定是1。当$access(y), splay(y)$以后，x一定是y的左儿子，直接双向断开连接

```
inline void cut(int x,int y){
    split(x,y);
    f[x]=c[y][0]=0;
```

```
    pushup(y);//少了个儿子，也要上传一下
}
```

那如果不一定存在该边呢?
充分利用好Splay和LCT的各种基本性质吧!
正确姿势——先判一下连通性（注意$findroot(y)$以后x成了根），再看看x,y是否有父子关系，还要看y是否有左儿子。
因为$access(y)$以后，假如y与x在同一Splay中而没有直接连边，那么这条路径上就一定会有其它点，在中序遍历序列中的位置会介于x与y之间。
那么可能y的父亲就不是x了。
也可能y的父亲还是x，那么其它的点就在y的左子树中

只有三个条件都满足，才可以断掉。

```
inline bool cut(int x,int y){
    makeroot(x);
    if(findroot(y)!=x||f[y]!=x||c[y][0])return 0;
    f[y]=c[x][1]=0;//x在findroot(y)后被转到了根
    pushup(x);
    return 1;
}
```

如果维护了$size$，还可以换一种判断

```
inline bool cut(int x,int y){
    makeroot(x);
    if(findroot(y)!=x||sz[x]>2)return 0;
    f[y]=c[x][1]=0;
    pushup(x);
    return 1;
}
```

解释一下，如果他们有直接连边的话，$access(y)$以后，为了满足性质1，该Splay只会剩下x,y两个点了。
反过来说，如果有其它的点，$size$不就大于2了么?

其实，还有一些LCT中的Splay的操作，跟我们以往学习的纯Splay的某些操作细节不甚相同。
包括$splay(x),rotate(x),nroot(x)$（看到许多版本LCT写的是$isroot(x)$，但我觉得反过来会方便些）
这些区别之处详见下面的模板题注释。

update（关于findroot中pushdown的说明）

蒟蒻真的一时没注意这个问题。。。。。。Splay根本没学好
找根的时候，当然不能保证Splay中到根的路径上的翻转标记全放掉。
所以最好把pushdown写上。
Candy巨佬的总结对pushdown问题有详细的分析
只不过蒟蒻后来经常习惯这样判连通性（我也不知道怎么养成的）

```
makeroot(x);
if(findroot(y)==x)//后续省略
```

这样好像没出过问题，那应该可以证明是没问题的（makeroot保证了x在LCT的顶端，access(y)+splay(y)以后，假如x,y在一个Splay里，那x到y的路径一定全部放完了标记）
导致很久没有发现错误。。。。。。
另外提一下，假如LCT题目在维护连通性的情况中只可能出现合并而不会出现分离的话，其实可以用并查集哦！（实践证明findroot很慢）
这样的例子有不少，比如下面“维护链上的边权信息”部分的两道题都是的。
甚至听到Julao们说有少量题目还专门卡这个常数。。。。。。XZY巨佬的博客就提到了

update（关于pushdown的说明）

我pushdown和makeroot有时候会这样写，常数小一点

```
void pushdown(int x){
    if(r[x]){
        r[x]=0;
```

```
        int t=c[x][0];
        r[c[x][0]=c[x][1]]^=1;
        r[c[x][1]=t]^=1;
    }
}
void makeroot(int x){
    access(x);splay(x);
    r[x]^=1;
}
```

这种写法等于说当x有懒标记时，x的左右儿子还是反的
~~那么如果findroot里实在要写pushdown，那么这种pushdown就会出现问题~~（参考评论区@zjp_shadow巨佬的指正）
再次update，蒟蒻发现这种问题还是可以避免的，若用这种pushdown，findroot这样写就好啦

```
inline int findroot(int x){
    access(x);splay(x);
    pushdown(x);
    while(lc)pushdown(x=lc);
    splay(x);
    return x;
}
```

~~当题目中维护的信息与左右儿子顺序有关的时候，pushdown如果用这种不严谨写法会是错的（比如[NOI2005]维护数列（这是Splay题）和洛谷P3613 睡觉困难综合征）~~
再次update，夏丶沐瑾巨佬指出这种问题也是可以避免的，把pushup这样写就好啦

```
inline void pushup(int x){
    pushdown(lc);pushdown(rc);//加上两个
    //......
}
```

所以此总结以及下面模板里的pushdown，常数大了一点点，却是更稳妥、严谨的写法

```
//pushr同上方makeroot部分
void pushdown(int x){
    if(r[x]){
        if(c[x][0])pushr(c[x][0]);//copy自模板，然后发现if可以不写
        if(c[x][1])pushr(c[x][1]);
        r[x]=0;
    }
}
void makeroot(int x){
    access(x);splay(x);
    pushr(x);//可以看到两种写法造成makeroot都是不一样的
}
```

这种写法等于说当x有懒标记时，x的左右儿子已经放到正确的位置了，只是儿子的儿子还是反的
那么这样就不会出问题啦
两种写法差别还确实有点大呢

模板

洛谷P3690 【模板】Link Cut Tree （动态树） （点击进入题目）
最基本的LCT操作都在这里，也没有更多额外的复杂操作了，确实很模板。

```
#include<bits/stdc++.h>
#define R register int
#define I inline void
#define G if(++ip==ie)if(fread(ip=buf,1,SZ,stdin))
#define lc c[x][0]
#define rc c[x][1]
using namespace std;
const int SZ=1<<19,N=3e5+9;
```

```
char buf[SZ],*ie=buf+SZ,*ip=ie-1;
inline int in(){
    G;while(*ip<'-')G;
    R x=*ip&15;G;
    while(*ip>'-'){x*=10;x+=*ip&15;G;}
    return x;
}
int f[N],c[N][2],v[N],s[N],st[N];
bool r[N];
inline bool nroot(R x){//判断节点是否为一个Splay的根（与普通Splay的区别1）
    return c[f[x]][0]==x||c[f[x]][1]==x;
}//原理很简单，如果连的是轻边，他的父亲的儿子里没有它
I pushup(R x){//上传信息
    s[x]=s[lc]^s[rc]^v[x];
}
I pushr(R x){R t=lc;lc=rc;rc=t;r[x]^=1;}//翻转操作
I pushdown(R x){//判断并释放懒标记
    if(r[x]){
        if(lc)pushr(lc);
        if(rc)pushr(rc);
        r[x]=0;
    }
}
I rotate(R x){//一次旋转
    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;//额外注意if(nroot(y
    if(w)f[w]=y;f[y]=x;f[x]=z;
    pushup(y);
}
I splay(R x){//只传了一个参数，因为所有操作的目标都是该Splay的根（与普通Splay的区
    R y=x,z=0;
    st[++z]=y;//st为栈，暂存当前点到根的整条路径，pushdown时一定要从上往下放标记
    while(nroot(y))st[++z]=y=f[y];
    while(z)pushdown(st[z--]);
    while(nroot(x)){
        y=f[x];z=f[y];
        if(nroot(y))
            rotate((c[y][0]==x)^(c[z][0]==y)?x:y);
        rotate(x);
    }
    pushup(x);
}
/*当然了，其实利用函数堆栈也很方便，代替上面的手工栈，就像这样
I pushall(R x){
    if(nroot(x))pushall(f[x]);
    pushdown(x);
}*/
I access(R x){//访问
    for(R y=0;x;x=f[y=x])
        splay(x),rc=y,pushup(x);
}
I makeroot(R x){//换根
    access(x);splay(x);
    pushr(x);
}
int findroot(R x){//找根（在真实的树中的）
    access(x);splay(x);
    while(lc)pushdown(x),x=lc;
    splay(x);
    return x;
}
I split(R x,R y){//提取路径
    makeroot(x);
    access(y);splay(y);
}
```

```
I link(R x,R y){//连边
    makeroot(x);
    if(findroot(y)!=x)f[x]=y;
}
I cut(R x,R y){//断边
    makeroot(x);
    if(findroot(y)==x&&f[y]==x&&!c[y][0]){
        f[y]=c[x][1]=0;
        pushup(x);
    }
}
int main()
{
    R n=in(),m=in();
    for(R i=1;i<=n;++i)v[i]=in();
    while(m--){
        R type=in(),x=in(),y=in();
        switch(type){
        case 0:split(x,y);printf("%d\n",s[y]);break;
        case 1:link(x,y);break;
        case 2:cut(x,y);break;
        case 3:splay(x);v[x]=y;//先把x转上去再改，不然会影响Splay信息的正确性
        }
    }
    return 0;
}
```

分类: 数据结构——链剖——LCT , 数据结构——平衡树——Splay , OI——算法总结 , OI——题解

标签: LCT , Splay

« 上一篇： 洛谷P2633 Count on a tree（主席树，倍增LCA，树上差分）
» 下一篇： 洛谷P1501 [国家集训队]Tree II（LCT,Splay）

评论列表

#51楼 2018-12-25 08:31 **✡smy✡**

@ M_sea
M_sea!!!!!!!!!!!!!!

捕捉

#52楼 2018-12-25 08:32 **M_sea**

@ Flash_Hu
因为smy在fAKe qwq

#53楼 2018-12-25 08:32 **M_sea**

@ smyjr
您fAKe死了