link-cut tree

分类： 数据结构 | 标签: 动态树,lct,splay | 作者： tgop_knight 相关 | 发布日期 : 2015-03-24 | 热度 : 931°

自己简单研究了下动态树，就先来一发博客吧

动态树

其实很多人都把LCT和动态树搞混了，所以我们先阐明一下定义。

动态树问题

——就是会动的树上的问题(⊙v⊙)

会动——就是树的形态或权值是会变化的

常见问题形势——

1. 维护两点的连通性
2. 维护两点路径权值的和，最大值，最小值，自定义运算值......
3. 维护LCA，直径，重心，自定义点与点的关系......

烦人的动态操作——

1. 连接两个点(先前是不连通的)
2. 把某个点变成某个点的父亲(先前是不连通的)
3. 断开两个点的连接(先前是联通的)
4. 修改点权，边权等等......

嗯，这种题一看就是裸数据结构对吧

所以就有了Tarjan发明的link-cut tree(OrzTarjen爷)

link-cut tree基础实现

什么是link-cut tree呢？

这个东西早在杨哲的集训队作业中提出了：QTREE解法的一些研究

上面的文段有很多正确且严谨的证明，下面不会再给出(OrzOrz)

如果通过上面的文段你已经理解了的话，就可以直接开始做题了！(Orz)

如果我没看懂呢？

没关系，你只需要两个知识：树链剖分、splay

首先我们需要了解什么是树链剖分

不会的话先看看吧， 有益身心健康 ：树链剖分讲解版

(ps:这个博主很爷但是很奇怪，大家要小心)

splay的文章我就不引用了，大家随便百度看书都有

说白了，LCT就是用splay实现动态树的链剖分

我们会发现，树链剖分是用的线段树(至少上面那个是)

可是动态树就是因为会动而讨厌

嗯，如果我们把用线段树维护的东西改成用伸展树，以深度为顺序构建splay

听起来可行的样子

多么和谐！

可是具体怎么实现呢？

膜拜一下树链剖分，我们学着用splay维护每次访问的点到它所在树根的链

我们定义 u_l 的儿子中访问时间最后的点为 w_l，若 u_l的中以儿子 v_l 为根的子树包含了 w_l，则称 v_l 为 u_l的

preferred child(就好像重儿子一样)，自然，我们可以脑补出以下概念:

preferred road(u与v的连边)。

preferred path(连续的preferred road所表示的一条路径)

剩下的非preferred child的儿子吗，我们就像轻儿子一样，直接记个path father就好

这里注意一下，path father只能表示两颗splay之间的关系，不能表示两点的关系(通过下面的图你可以明白)

嗯，想像一下，一棵树原本被分成多个线段树，这些线段树又像树一样相连。

唰！线段树变成了splay!

听起来挺不错的样子是吧(⊙ o ⊙)

接下来怎么办？

别急，我们简单梳理一下

没错，我们把一颗树用splay剖分了

每颗splay对应着一条路径(preferred path)

(ps：我记得杨哲的paper里说的好像是个 aux什么什么 tree来着，我这里就直接用preferred path代替了)

splay之间构成了一片森林

这样的话原本要维护的森林就可以直接还原

问题来了

怎么取出某个点到所在树根的路径呢(⊙o⊙)？

我们要把 x 到根的路径变成一条preferred path！

这就用到了 access 操作

ps：请大家在下面务必分清原树和路径(preferred path)

嗯哼！

我们来看看 access 的操作吧

```
void access(int x)
{
    splay(x);//把x splay到所在preferred path的根
    cutright(x);//切断x与其右子树的连边
    while (splay[x].pf)//若x还存在path father
        {
        int u=splay[x].pf;
        splayup(u);//把u splay到所在preferred path的根
        cutright(u);//切断u与其右子树的连边
        splay[u].r=x;
        splay[x].pf=0;
        splay[x].f=u;//将u的右儿子置为x
        update(u);//更新u
        x=u;
        }
    return ;
}
```

(一头雾水(⊙o⊙)…)

这是干嘛呢？

这就是把 x 到所在树的树根变成preferred path

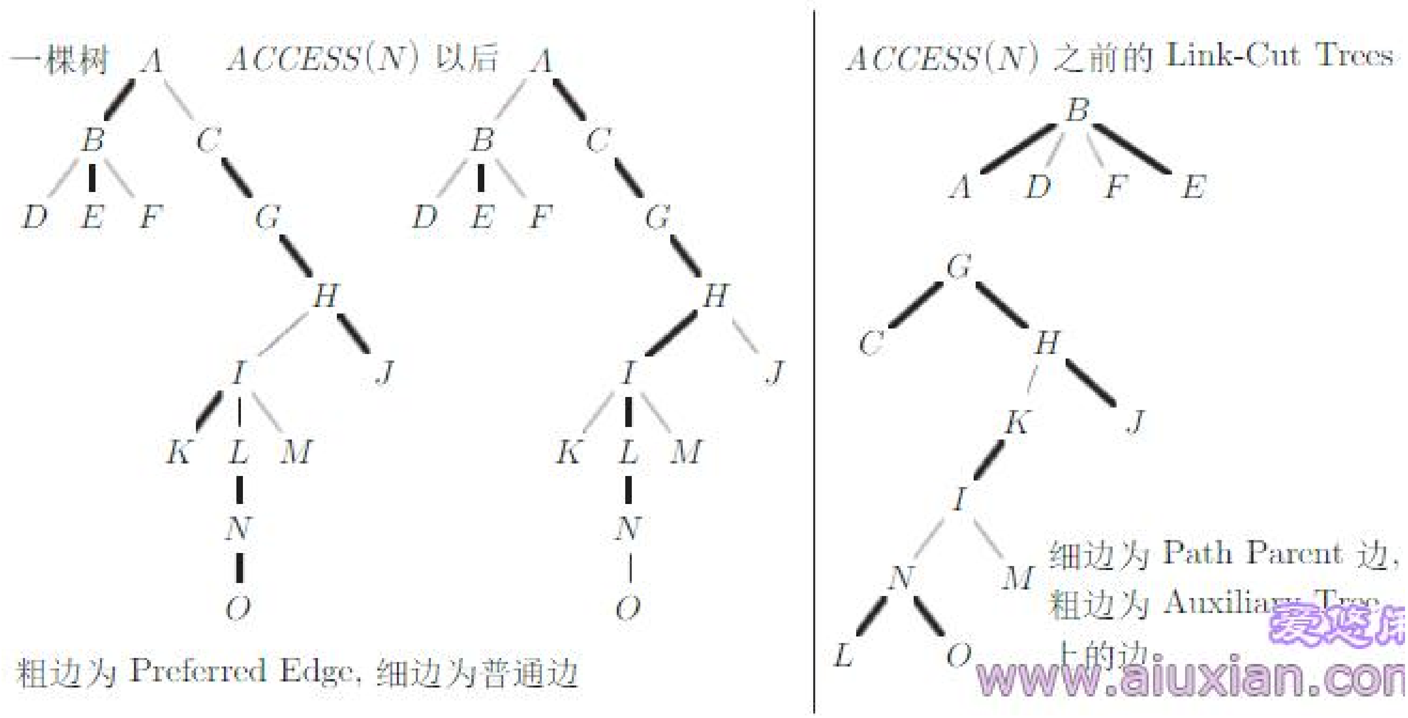
其实实现挺直接的，就是利用了splay的伸展性，无脑向上，splay不行就走path father，在路程中顺便更新下preferred path

实现就这几行

这里注意一下：

cutright操作只是在preferred path中断开，并不是在原树中断开(也就是说断开后，你要把右儿子的path father置为x)

下图为 Link-Cut Trees 的一个结构示意图，及一次 ACCESS 操作的前后对比图.



嗯我是盗图狗 ，右边那个A和G之间好像少了点啥……

不明觉厉

通过splay的伸展性，切断再重连得到新的preferred path(Tarjan我真的好崇拜您Orz)

这就是 access

诶，说这些有啥用？

其他的怎么维护呢(⊙v⊙)？

有了 access，我们可以做如下操作

1.断开 x 与其父亲的连边。

代码如下

```
void cut(int x)
{
    access(x);
    splay(x);//取出x到根的路径并将x变成路径根
    int l=splay[x].l;
    splay[x].l=0;//删除x的左孩子
    if (l)//如果左孩子存在，即x不为树根
        {
        splay[l].f=0;//断开左孩子和x的连边
        splay[l].pf=0;//包括path father也要清零(虽说其本来就是0)
        update(x);
        splay(l);//由于l变成了一棵树的根，所以splay它
        }
    return ;
}
```

相信通过代码和解释应该都能搞清楚怎么弄的了(^o^)/~

我们的splay是以深度为关键字的

如果是 x 和 y 的连边呢？

你可以稍微维护一下每个节点的父亲

通过 access(x)和 access(y)来区分我也不拦着你

2.判断 x 与 y 是否在同一棵树中

我们只需判断 x 和 y 所在的根就可以了

$access(x)$之后直接找最左(深度最小)的点就是啦O(∩_∩)O~~！

代码如下

```
int root(int x)
{
    access(x);
    splay(x);//取出x到根的路径并将x变成路径根
    while (splay[x].l) x=splay[x].l;//无脑找最左
    splay(x);//因为找出来的是树根，所以splay一发
    return x;
}
```

这里我还是给出找爸爸的代码：

```
int father(int x)
{
    access(x);
    splay(x);//取出x到根的路径并将x变成路径根
    x=splay[x].l;
    while (splay[x].r) x=splay[x].r;//无脑找最右
    return x;
}
```

3.将 x 变成所在树的树根

代码很简单

```
void beroot(int x)
{
    access(x);
    splay(x);//说多了不想说了，以后的都直接省略
    splay[x].rev^=1;//给x所在路径打上翻转标记
    return ;
}
```

诶诶诶(O_O)？

这个翻转标记匪夷所思啊！

仔细一想，我们不过是将整条路径翻转了一下

路径是从 x 到根

翻转一下不就变成根到 x 了？

多么机智是不是^_^？

4.在 x 与 y 中连接一条边(这里是把 y 变成 x 的子树)

嗯，首先 $root(x)$ 不等于 $root(y)$

接着我们将 y 变成所在子树的根

把 y 的path father置为 x

代码：

```
void join(int x,int y)
{
    int rootx=root(x),rooty=root(y);
    if (rootx==rooty)
        {
        //printf("加了就不是树了！！！");
        return ;
        }
    access(x);
    beroot(y);//变成根
    splay[y].pf=x;//接上x和y
    return ;
}
```

5.取出某条路径 x 到 y 的所查询权值

首先这条路径要有($root(x) == root(y)$)！(ps 在程序中我不会给出)

按照常理，我们先求LCA

先 $access(x)$

再 access(y)

将 xl 所在preferred path的path father找出来，就是LCA

如果没有path father，则为 xl 与 yl 之间的一个

如果LCA就是根，那么只需 access(x)、 access(y)即可

否则我们先 access(x)，再 access(LCA的父亲)

那么LCA所在preferred path就是 xl 到LCA的路径

yl 类似

计算的时候别忘了对LCA的重复判断

代码

```
int LCA(int x,int y)
{
    access(x);
    access(y);
    splay(x);
    splay(y);//注意一下splay是放在两个access后
    if (splay[x].pf)
        return deep[x]<deep[y]?x:y;//深度小的是LCA
    else
        return splay[x].pf;
}

int find(int x,int LCA)
{
    access(x);
    access(father(LCA));
    splay(LCA);//把LCA变成所在路径的根
    return splay[LCA].所需权值;
}
```

等等！

思考如下方法：

我们先将 xl 变成整棵树的树根

再 access(y)

如果是有根树我们在做完操作之后再把根变回去

这样，LCA就被我们完美避开了

代码：

```
int find(int x,int y)
{
    beroot(x);
    access(y);
    splayup(y);
    return splay[y].所需权值;
}
```

突然一下简洁了许多！(^o^)/~

程序不就是追求这个么？

好了，到这里基本的LCT你其实已经会了Y(^o^)Y

在练手之前，我提醒一下几点：

1.注意这颗splay是带区间翻转操作的

2.如果维护的值较多，请仔细理清权值之间的关系并且固定好顺序

嗯，练手题：

spoj 375 QTREE

这题是出了名的

维护一棵树支持两个操作

1.改变第 i 条边的权值

2.询问 a 到 b 的路径上的边权最大值

(ps：spoj上的QTREE系列题目都可以做一做)

山东省选2008 洞穴探测cave

维护一棵树支持三个操作

1.在 x 和 y 之间连一条边

2.将 x 与 y 的连边断开

3.询问 x 与 y 是否联通

输入数据保证操作合法

hdu4010

维护一棵树支持四个操作

1.在 x 与 y 之间连一条边

2.将 x 作为根，然后断开 y 与父亲的连边

3.将 x 到 y 的路径上所有点点权 $+k$

4.询问 x 到 y 的路径上的点权最大值

每次操作若不合法请输出 -1

如果你能一遍A这三题，说明你的LCT已经上路了(๑´ㅂ`๑)

一点使用技巧

一、边权的维护技巧

之前的题目都没有涉及到边权啊，失误失误(⊙o⊙)！

关于边权的维护，很多人会这么想：

反正一颗节点为 w 的树最多就 $w-1$ 条边

嗯，除了根之外，其他的点全部捆绑上自己与父亲的连边进行维护

多么简单？

一点也不好么╮(╯_╰)╭

如果你去写写你就知道了

$access$会出大问题

1.path parent的修改会涉及到这条边的边权

2.当一条preferred path的根并不是这条链的顶端的时候，你会发现这个根绑定的权值和它的path parent的权值都需要维护(稍微不小心就会写错)

3.如果你不相信，你可以自己去实现实现，如果有简单的实现代码，请务必拿来让我膜拜一下/Orz

如果 $access$出了问题，那么整个LCT就变得不好了╮(╯﹏╰）╭

嗯，其实我一开始也不知道怎么改进，后来我膜拜了其他Orz的代码

“把一条边拆成一个点，然后按照点维护不就好了么(ーー゛)？”大神这么说

怎么就这么机智呢？还是要Orz一下

来道题练练手吧：

NOI2014 魔法森林forest

在一个 n 个点的无向图上，每条边 i 有两个边权 a_i, b_i

请你找出一条从 1 到 n 路径，使得两个权值的最大值之和最小

啥？这真的是动态树(O_O)？ 博主你在逗我吧？

嗯(⊙v⊙)，这真的是动态树(虽然大牛们说自己都是写的spfa(ˇ_ˇ))

思考如下做法：
1.我们按照某一个边权 a_i 从小到大排序，将其依次加入图中
2.如果两点不连通，则直接加入
3.如果两点连通，则将两点原本的路径上的边权 b_i 最大的边找出来，然后与新加的边比较，如果新边较大，则不加，否则就把找出来的边删除，然后加入新边
4.如果成功加入某条边之后 1 与 n 联通，则统计答案

算法的思想大致如下：
总之就是按照 a_i 的边权排个序，轮流加入，维护图的边权b的最小生成树
很贪心对不对？
为什么能贪心呢？
你的问题实际上等价于：为什么 $kruskal$ 算法是对的
这个证明满大街都是，如果你会拟阵则更方便
(UOJ上你是可以看到AC的代码的，你可以膜拜，如果发现问题你也可以hack，嗯，总之UOJ大法好)

二、动态无向图的连通性维护

先来看道题：
上海省选2008 堵塞的交通traffic
题意我就不解释了，因为我并不会讲(这题正解是线段树)

考虑变式题，如果是一个无向图，这该怎么实现连通性的维护呢？
其实很简单
我们离线思考这个问题就会发现:
如果某个时刻两个点之间有多条路径，我们只需维护被破坏时间较后的那条路就可以了
也就是边权被破坏时间的最大生成树
嗯~(≧▽≦)/~，就是这样
如何维护最大生成树可以参考魔法森林的维护方法

妈妈如果我想维护最短路怎么办(=@__@=)？
这个......动态树真的可以做么(其实我也不知道)？
说不定要用到更高端的数据结构，说不定根本就不行
谁知道呢？
(翱犇告诉我：洗洗睡吧，好像没有有竞赛价值的做法)

一点总结

解决动态树的东西肯定是不止LCT这一个东西的，还有ETT(Eular-Tour Tree)
那么为什么非要讲这个呢？
当然是因为LCT是最容易实现的，也足够应付大部分动态树的题目了

额(⊙o⊙)...
如果你还不满足自己在动态树上的造诣的话
你可以去看看翱犇2014年的集训队论文《浅谈动态树的相关问题及简单拓展》(网上有2014集训队论文捆绑版)
如果你知道什么是仙人掌，你也可以去膜拜膜拜VFleaKing大神，并思考思考动态仙人掌(这个在UOJ和BZOJ上都有)

留下一个思考题吧，如果我们在对路径的权值修改操作的同时，加上对子树的权值修改操作，怎么办呢？