Link-Cut Trees

Link-Cut Trees 是由Sleator 和Tarjan 发明的解决这类动态树问题的一种数据结构[1]. 这个数据结
构可以在均摊$O(\log n)$ 的时间内实现上述动态树问题的每个操作.
下面我们先介绍Link-Cut Trees 的概念，然后介绍它的支持各种操作，最后对它的复杂度进行分析.

一: Link-Cut Trees 的定义

access:访问

[英]['ækses][美]['æksɛs]

preferred:优先

[英][prɪ'fəd][美][prɪ'fɜ:d]

path:路径

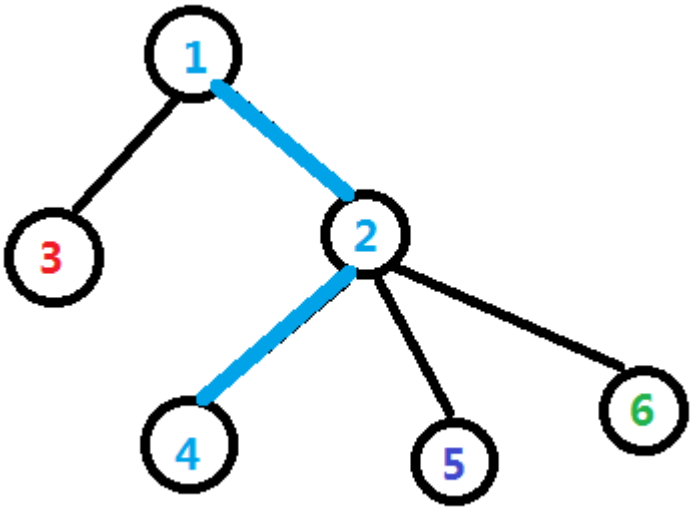
[英][pɑ:θ][美][pæθ, pɑθ]

auxiliary: 辅助

[英][ɔ:g'zɪliəri][美][ɔg'zɪljəri, -'zɪləri]

什么是 link-cut trees?

link-cut trees 是把树分解成多个树链，并且把每条树链都分别储存到一颗 splay 中。当我们需要对一颗树上的路径进行操作的时候，利用 splay 进行分离与合并，把这条树上的路径储存到同一棵 splay 中，然后再操作这颗 splay 就行了。



我英文不太好…下面我们把上述的英文概念换成中文的…

Preferred Child: 偏好子节点。

Preferred Edge: 偏好边。

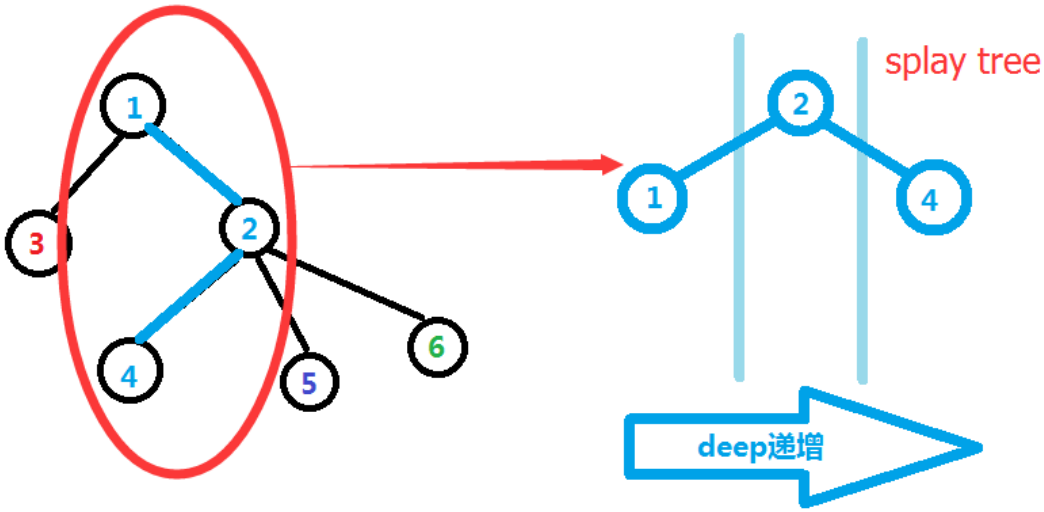
Preferred Path: 偏好路径。

Auxiliary Tree: 辅助树。

Path Parent: 路径父节点。

我们来举个例子说明一下这些概念，上图是一颗以 1 为根节点的树。1 的 Preferred Child 为 2,2 的 Preferred Child 为 4,3、4、5、6 没有 Preferred Child。

与树链剖分相同的是，link-cut trees 也是把一颗树变成了多个线型的链，但是 link-cut trees 采用了 splay tree 来储存这些链，方便后面的合并与拆分。

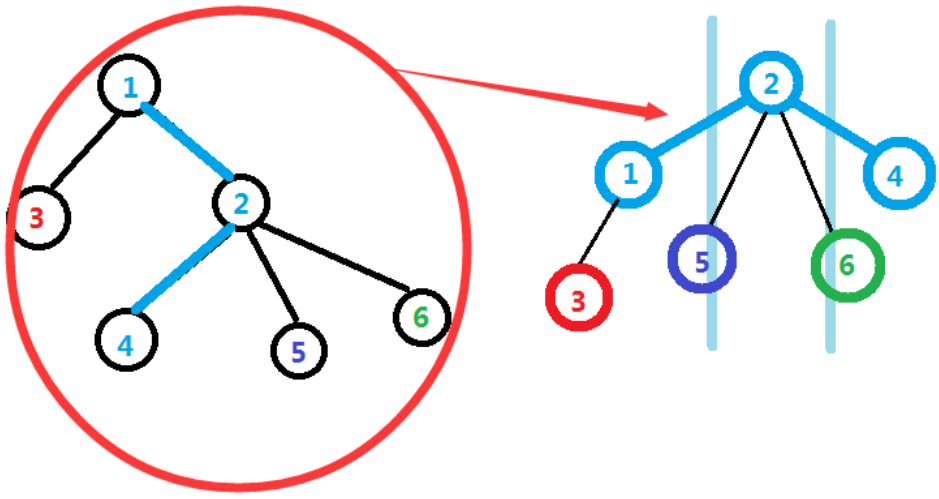


在这颗 splay 中，离根节点近的边被储存到 splay 的左边，离根节点较远的被储存到右边，splay tree 是以 deep 作为关键字排序的。

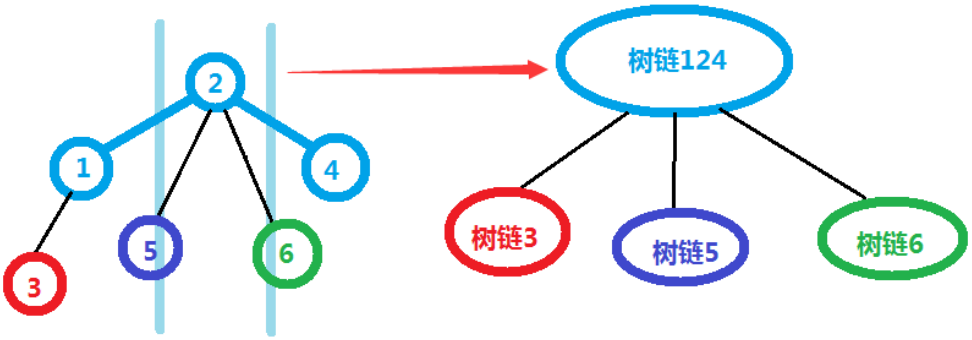
那么如何储存多条树链呢？

我们称 splay 的根节点的父亲为路径父节点，这个节点是这条树链中 deep 最小的点在原树的父节点。

右边的这颗树就是 link-cut trees。



如果把右边的 link-cut trees 抽象一下变成这样



把一条树链当成一个点，把轻边当成树上的边，我们可以的得到一颗虚树，很明显，这颗虚树无论宽度还是高度都没有最开始的树大，既使是一颗很大的树也可以压缩成一颗很小的虚树。当然…如果虚树的结构不再发生改变，在每一个节点都用线型的数据结构进行加速的话就能达到 log 级的效率（而且相当快），这个是树链剖分的原理，你可以把树链剖分理解成静态 link-cut tree，不过说起 link-cut tree 的原理…我觉得这点不是最主要的（事实上 link-cut tree 总是能在 log 级的时间内把需要查询的区间构造成主链，这点是最重要的）。

二：基本操作

一、access（访问）

ACCESS 操作是 Link-Cut Trees 的所有操作的基础．假设调用了过程 *ACCESS*(*v*)，那么从点 *v* 到根结点的路径就成为一条新的 Preferred Path.

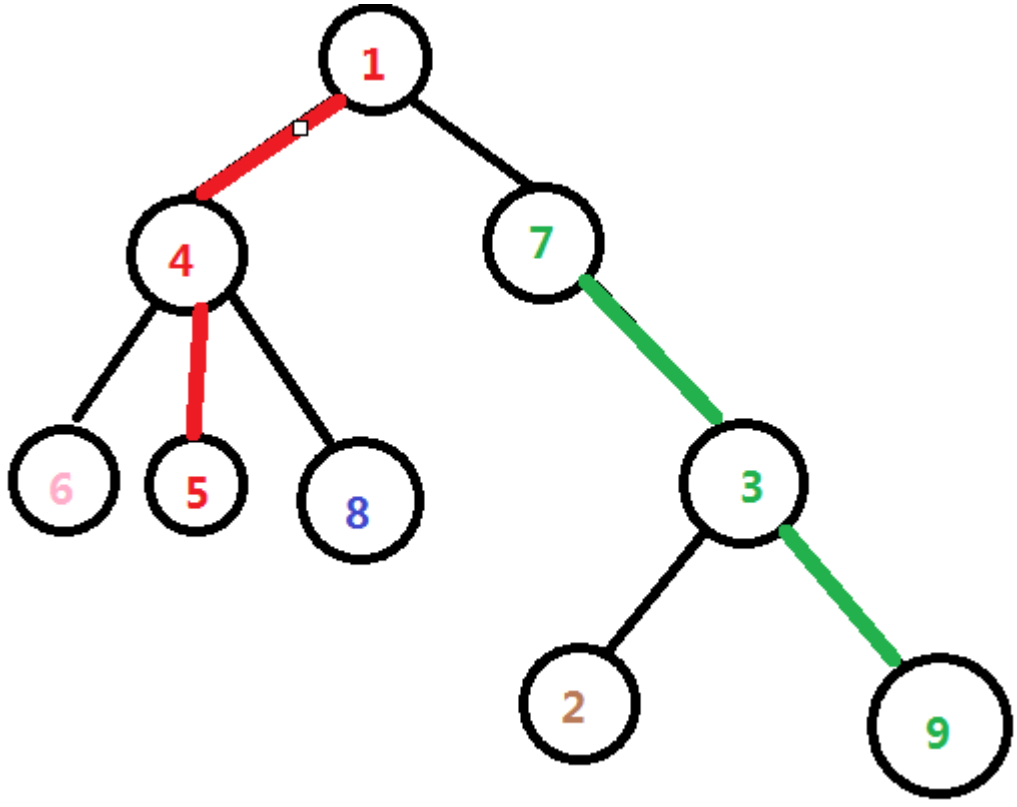
如果执行了一个点的 access 操作，那么从根节点到这个点路径上的所有的点就会变为偏好孩子，所有的边就会变为偏好边，这条路径也就变为一条偏好路径，同时，从根节点到该节点的这条路径就被储存到了 link-cut trees 根节点的 splay 中。

那我们要怎么做呢？涉及到 splay 的分离与合并岂不是很麻烦。

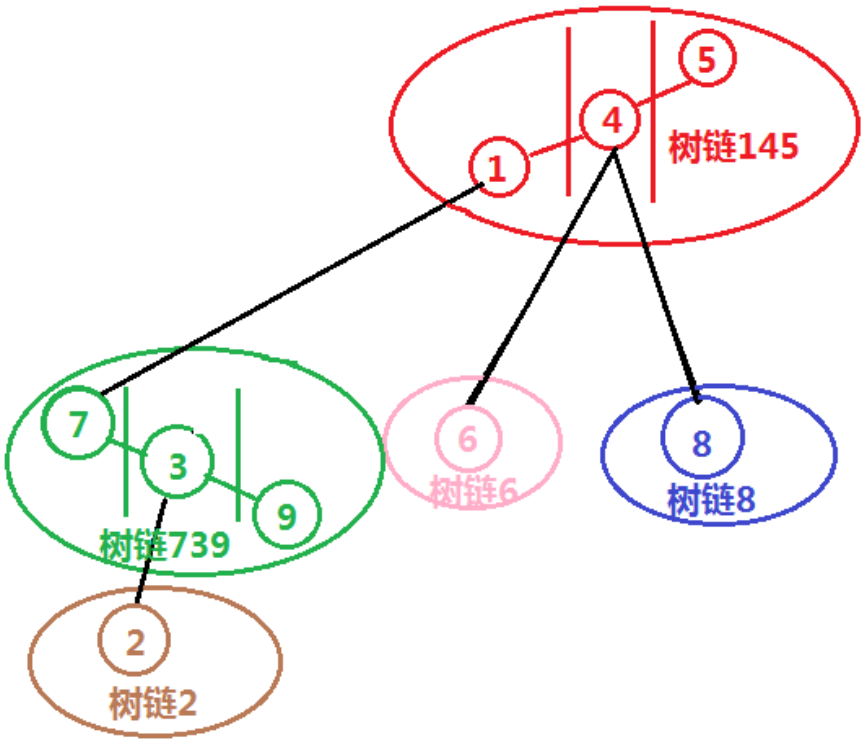
假设访问到 x 节点，我们先把 x 伸展到它所在的 splay 的根，然后断开它的右孩子，因为被访问到的点是没有偏好孩子的。这样 x 所在的 splay 就是一条从根的方向延伸过来的一条树链（当然，它并不完整）然后我们访问 x 所在的树链的路径父节点，并把它旋转到它所在 splay 的根，然后断开它的右孩子，并连接到 x 所在的树链，然后不断重复这一过程，最终，我们就会得到一条从根节点延伸到 x 的一条树链。

我们模拟一下过程。

比如现在树的模样是这个样子的。



它所对应的 link-cut tree 是这样的



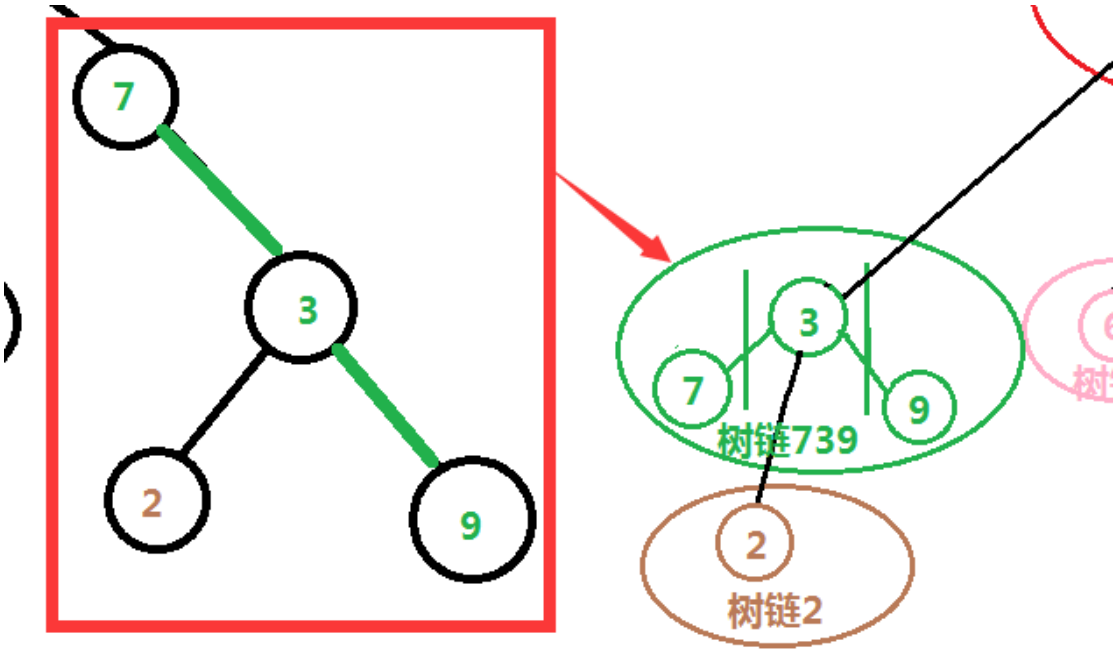
以 access(2)为例

先把 2 旋转到树链 2 的根，把 2 的右孩子设为 0。

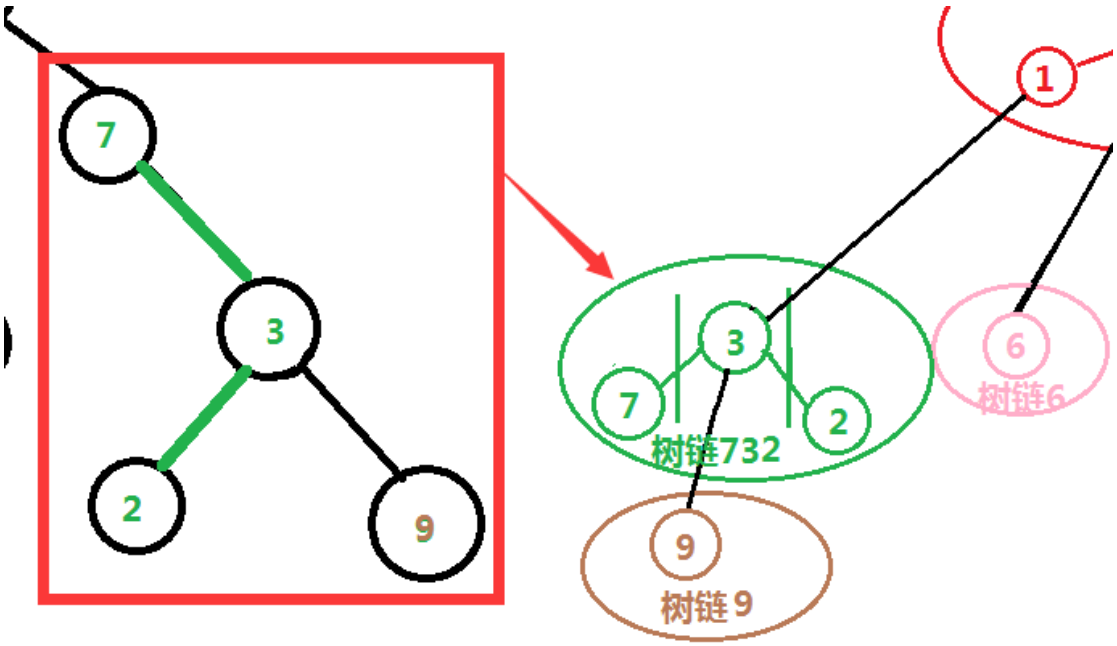
然后处理它的 Path Parent。

那么就处理 3，先把 3 旋转到根，把 3 的右孩子设为 2。（？为什么这么做？）

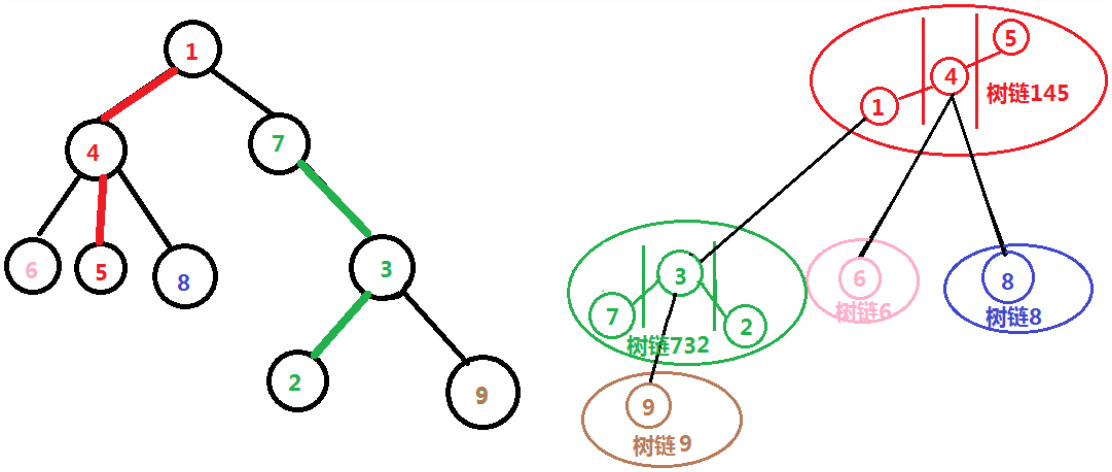
首先：根据 link-cut tree 的定义，辅助树中被放在右边的都是 deep 比较深的点，所以当我们把一个点旋转到辅助树的根部时，它的左右孩子分别为两条树链，左孩子链接的树链是从根部过来的，而右孩子链接的树链是到更深处的，所以我们要先把这个点旋转到根部，然后断开向深处延伸的树链，链接到需要访问的树链上。



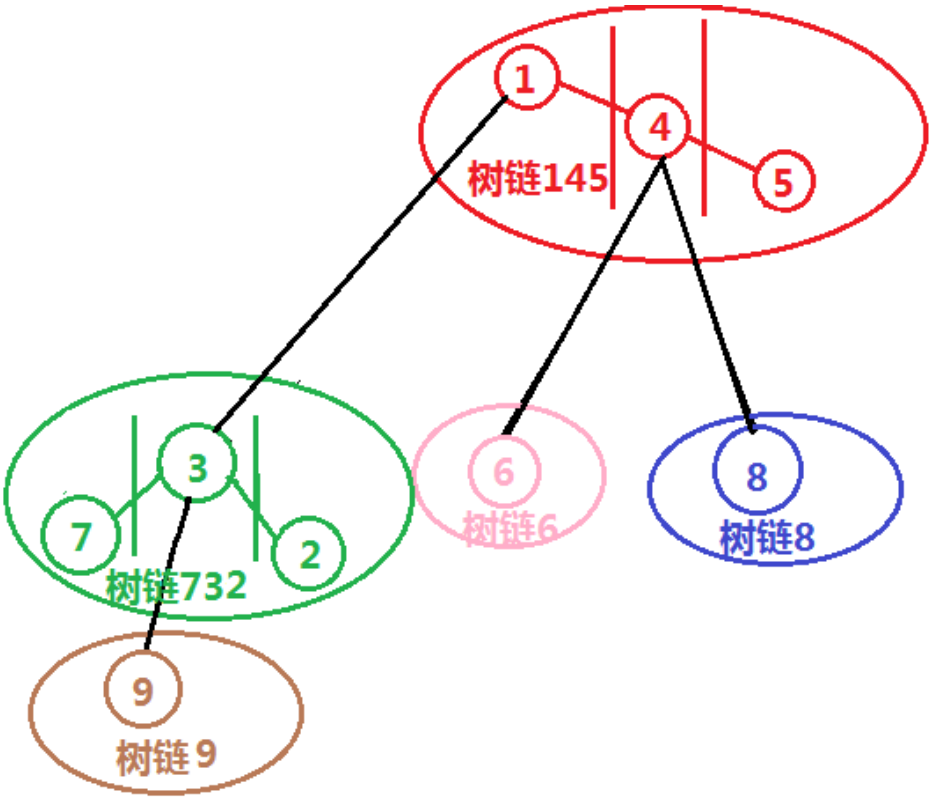
修改完成后将会变成这个样子：



接下来我们要访问 3 的 Path Parent，就是 1 啦，先把 1 旋转到它所在的辅助树的根部，断开它的右孩子，连接到 3 上。

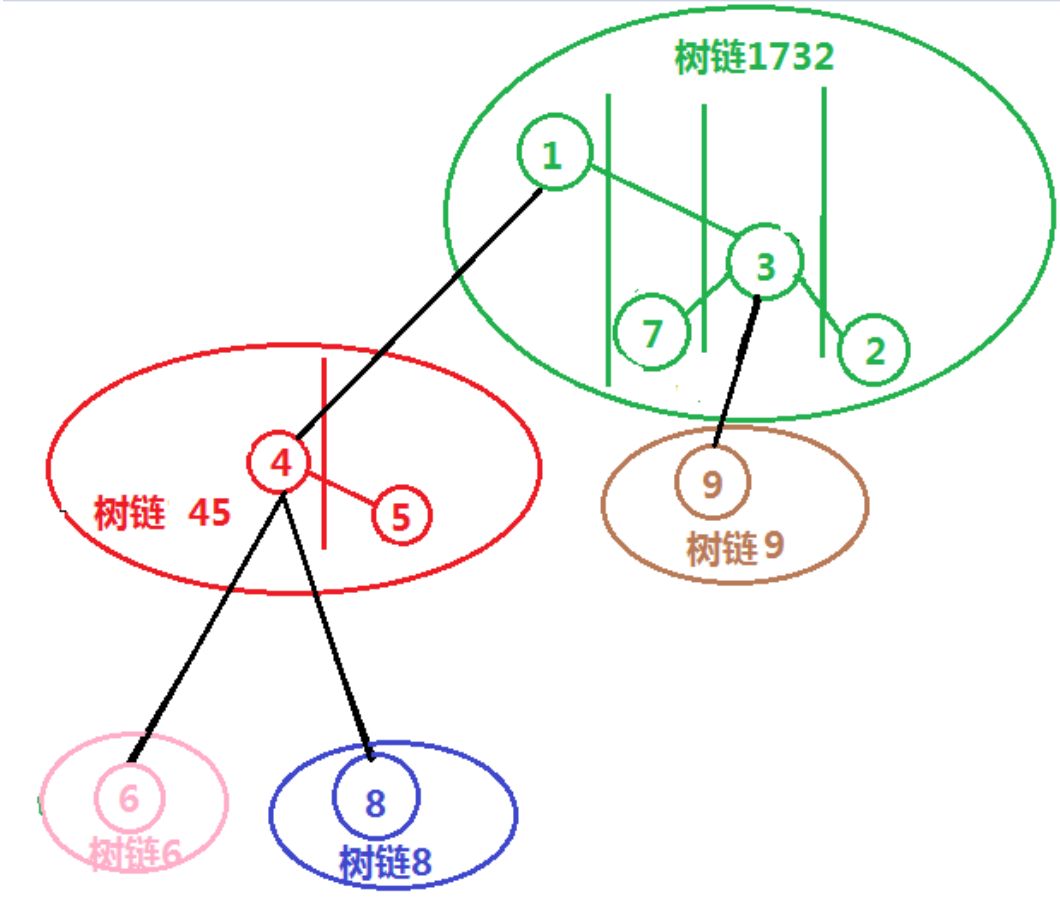


先 **splay(1)**。

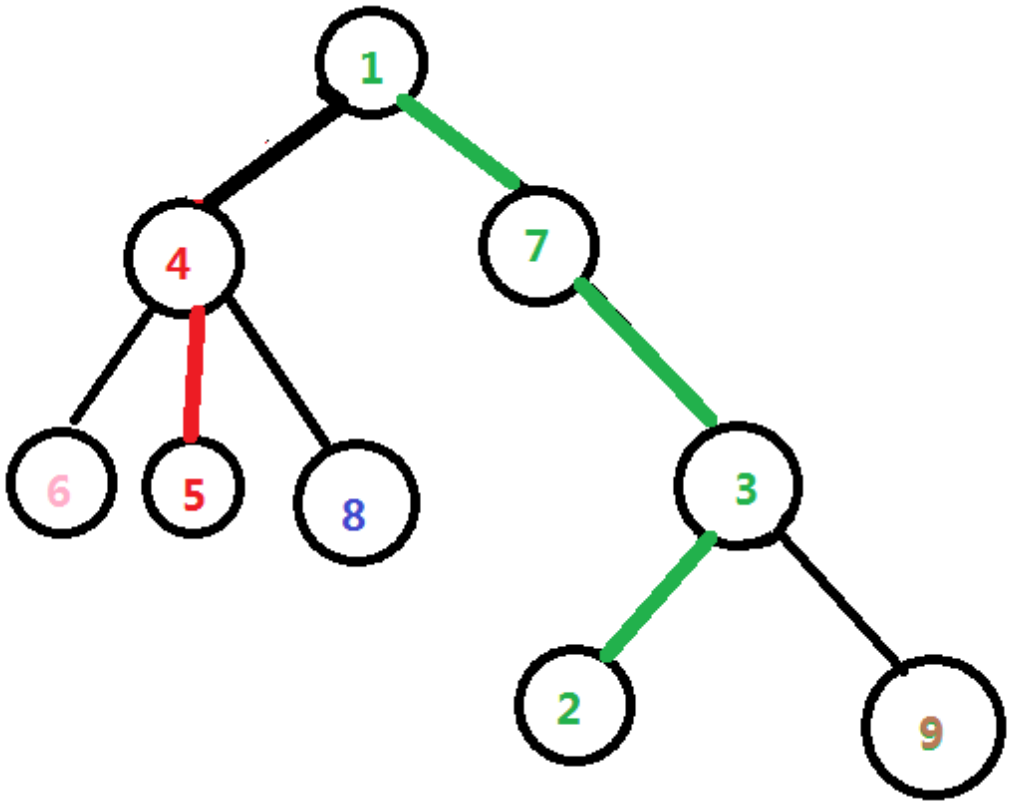


把 1 的右孩子断开接到 3 上面去。

做完这步操作 **link-cut tree** 的形状如下图所示



原树：



这样，我们就完成了 access(2)的操作。

具体代码实现：

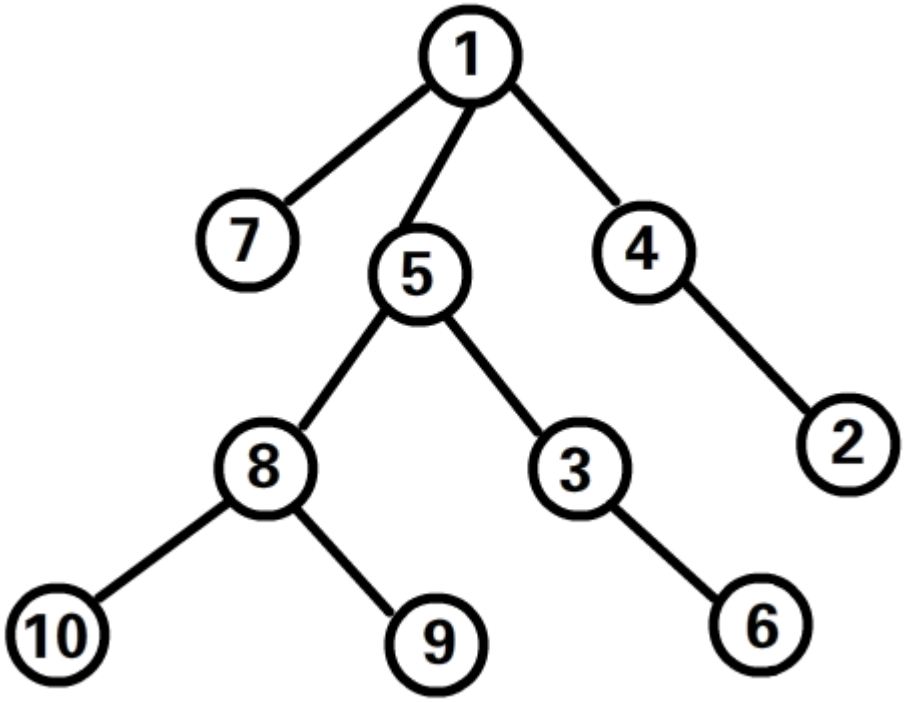
```
void access(int root)
{
    int temp=0;
    while(root)
    {
            splay(root);
            t[root].ch[1]=temp;
            updata(root);
            temp=root;
            root=t[root].fa;
    }
    return;

}
```

二、lca（如果不是直接问 lca 的话基本上用不着，如果是静态树就更用不着了，除非就是动态树直接问 lca 我们才用得着…）

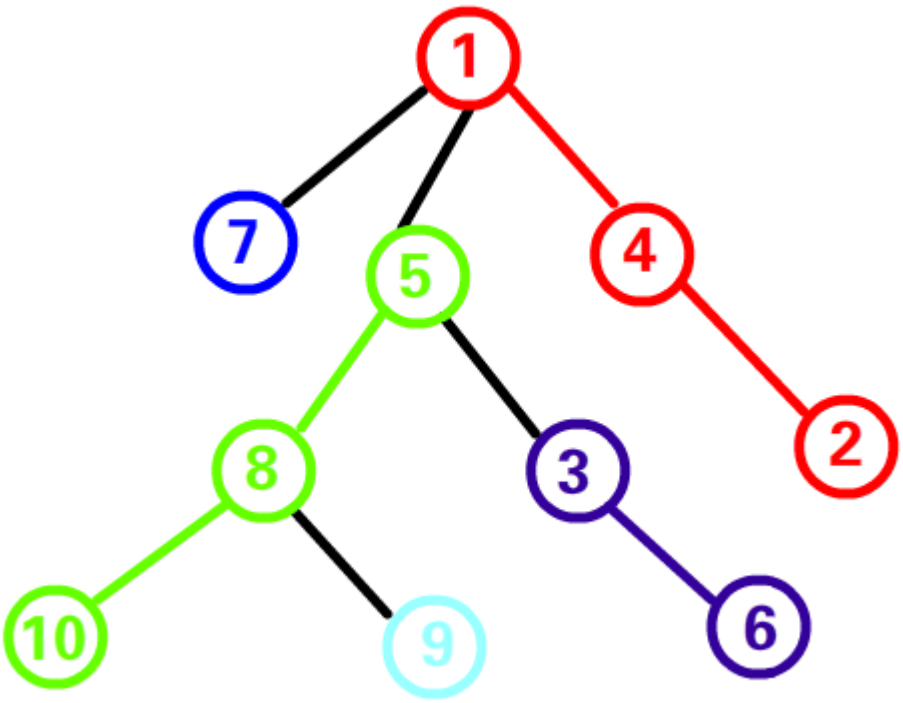
access 操作是 **link-cut tree** 一切操作的基础。

下面举个例子来说怎么求 **lca**。

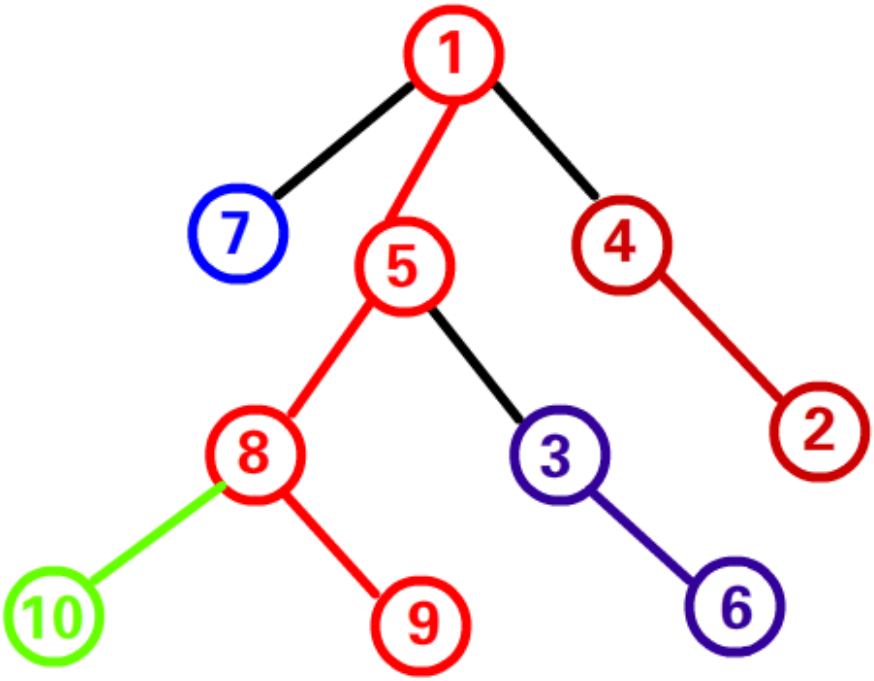
下面我给定这么一棵树。



当然，我们一开始用 **link-cut tree** 储存起来要把它分解成树链



我们先说原理，比方说我要求 9 和 6 的最近公共祖先，我们首先访问 9，这样就能构造出一条根节点到 9 的通路，如下：



如果我们接下来访问 **6**，那么最后一次需要修改右孩子的节点是几？是 **5——6** 和 **9** 的最近公共祖先。为什么呢？因为当我们从根节点画两条通路分别通向两个点时，这两条通路中总是会重合一部分，这些重合的部分在我们两次 **access** 操作中是不用变的，那么最后一次变化的点就是两点的最近公共祖先。一点都不魔法嘛。

然而接下来发生的事更加不魔法。

由于我们在 **link-cut tree** 中使用的基础数据结构是 **splay tree**，所以最后一次访问到的点…没错，被我们旋转到根节点了。

但是其实我们并不用查询根节点是谁…我们用代码实现来说明

首先我们要微调一下 **access** 操作

```
int access(int root)
{
        int temp=0;
        while(root)
        {
                splay(root);
                t[root].ch[1]=temp;
                temp=root;
                root=t[root].fa;
        }
        return temp;
}
```

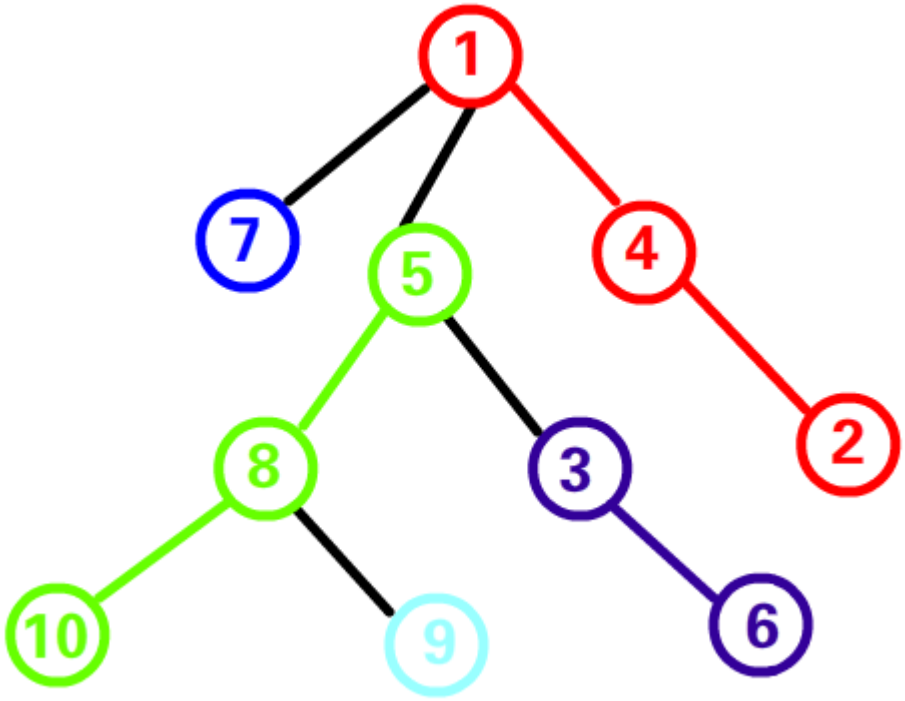
我们让 **access** 操作每次返回 **temp**（最后一次修改右孩子的点）。

那么 **lca** 怎么写…就不用我说了吧

```
int lca(int root1,int root2)
{
    access(root1);return access(root2);
}
```

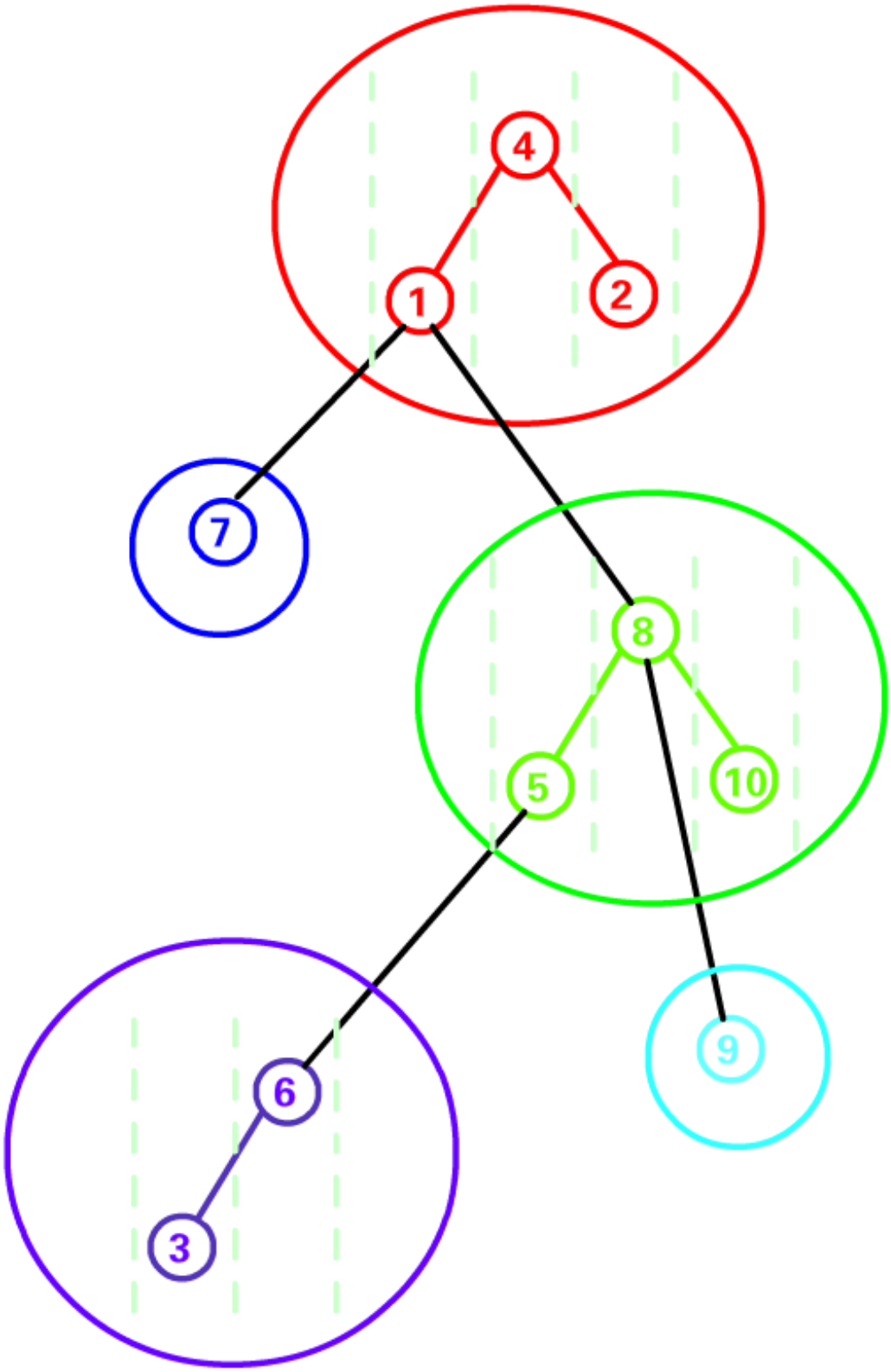
我们来真正模拟一下求 **lca** 的过程!

现在原树是这样的~



然而对应的 **link-cut tree**…容我先画会图去。

（十年后…）


4
1
2
7
8
5
10
9
6
3

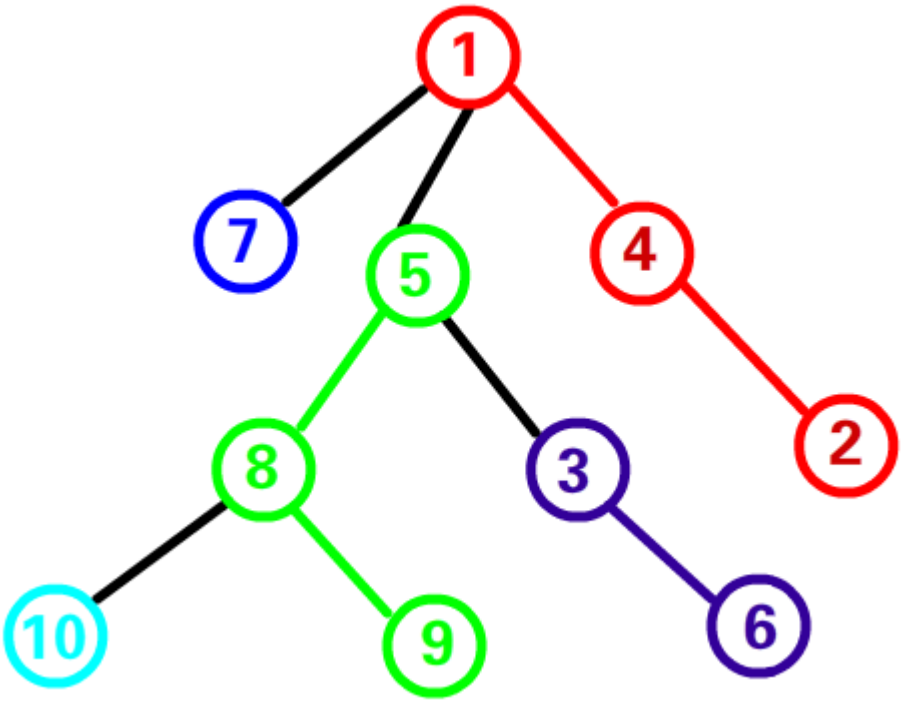
那么首先执行 **access(9)**。

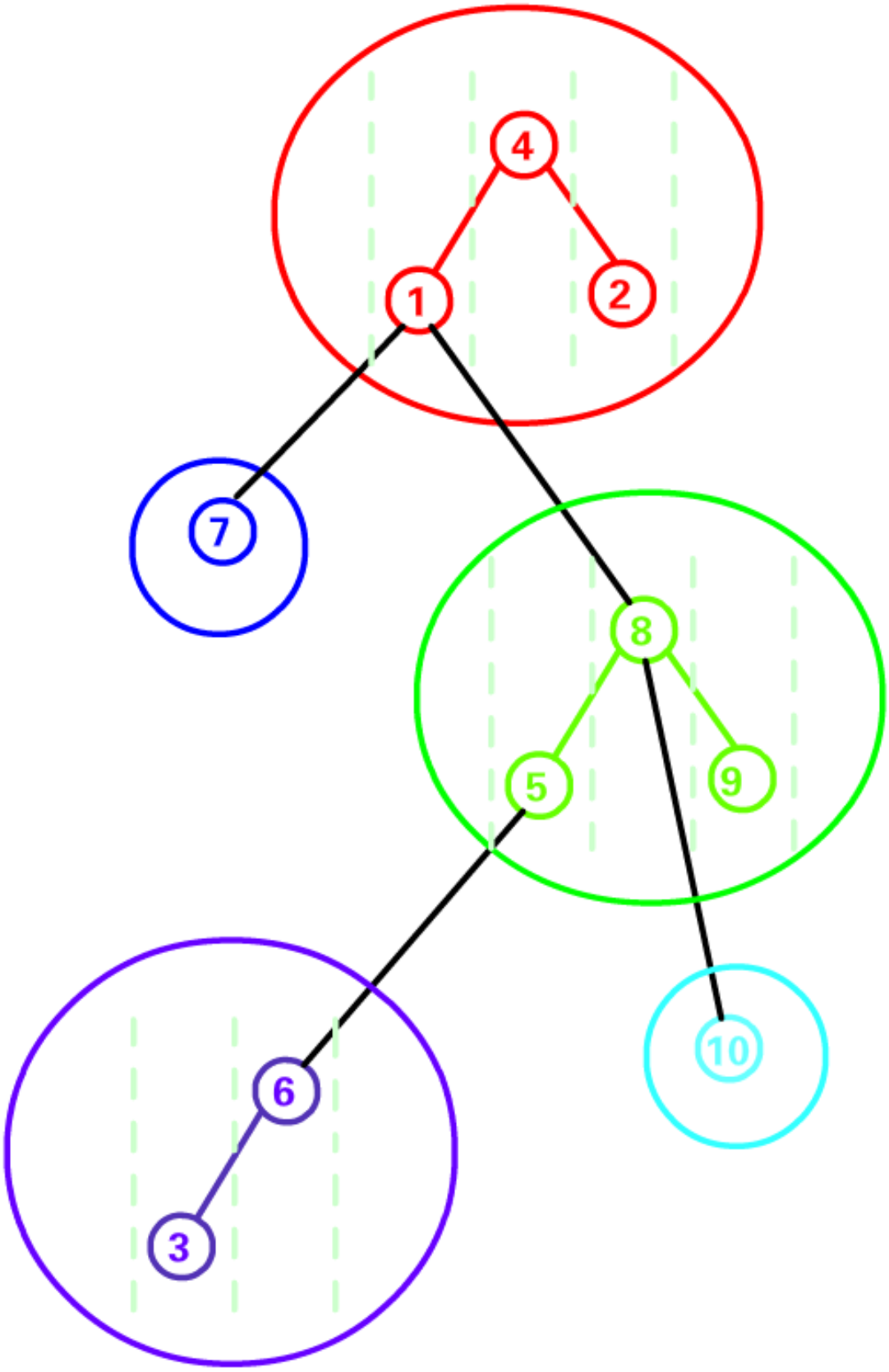
则先 **splay(9)**，把 9 的右孩子改为 0。

然后没有什么变化。

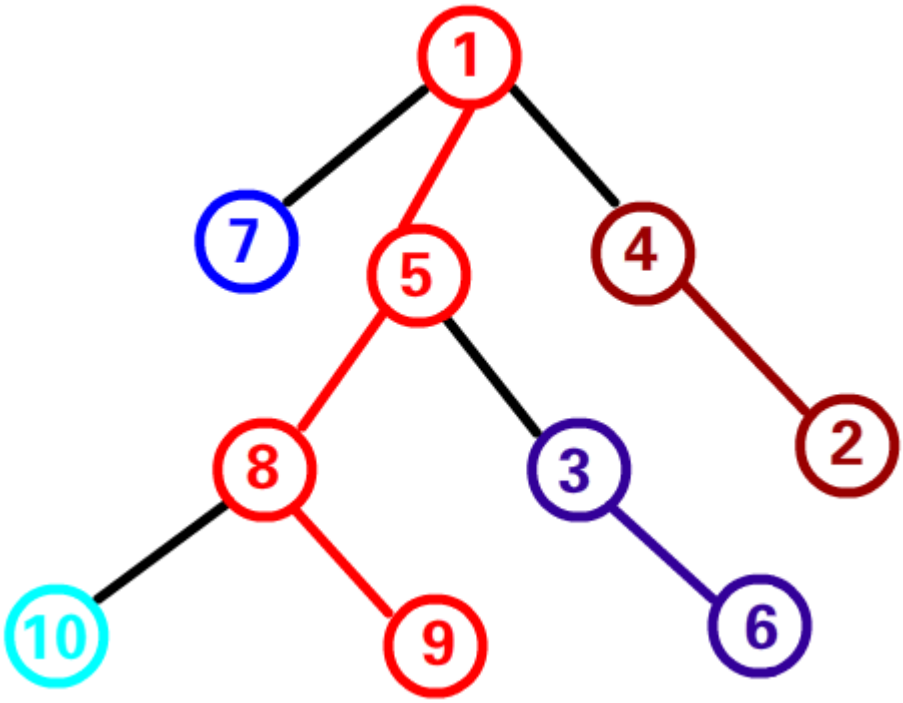
接下来 **splay**（8），把 8 的右孩子改为 9。

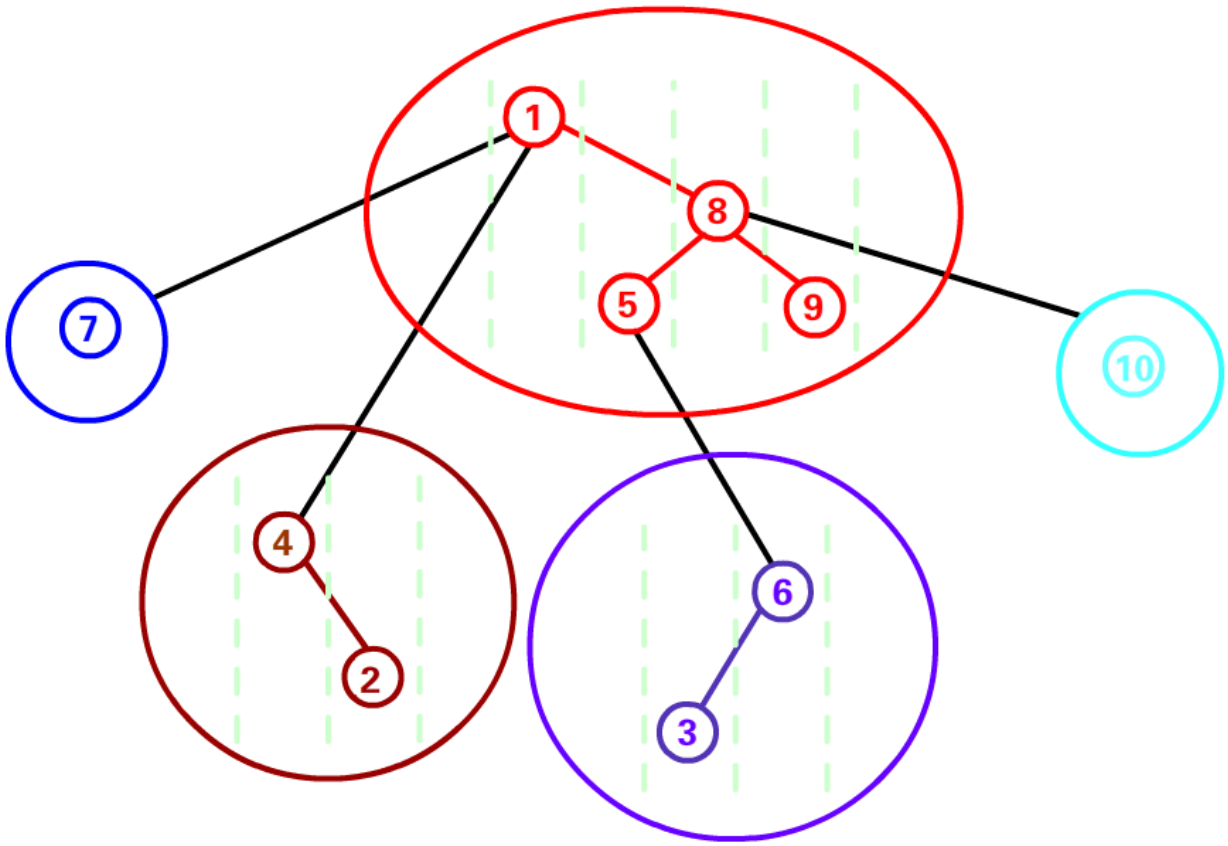
变化如下：




4
1
2
7
8
5
9
6
10
3

接下来 **splay(1)**，把 **1** 的右孩子改为 **8**，变化如下：





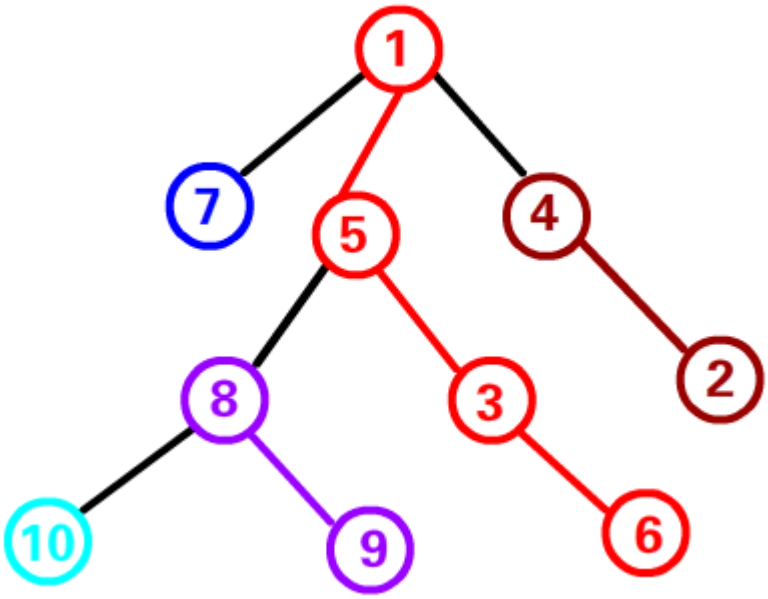
好的~我们现在做完了 access(9)的操作，接下来然我们进行 access(6)的操作。

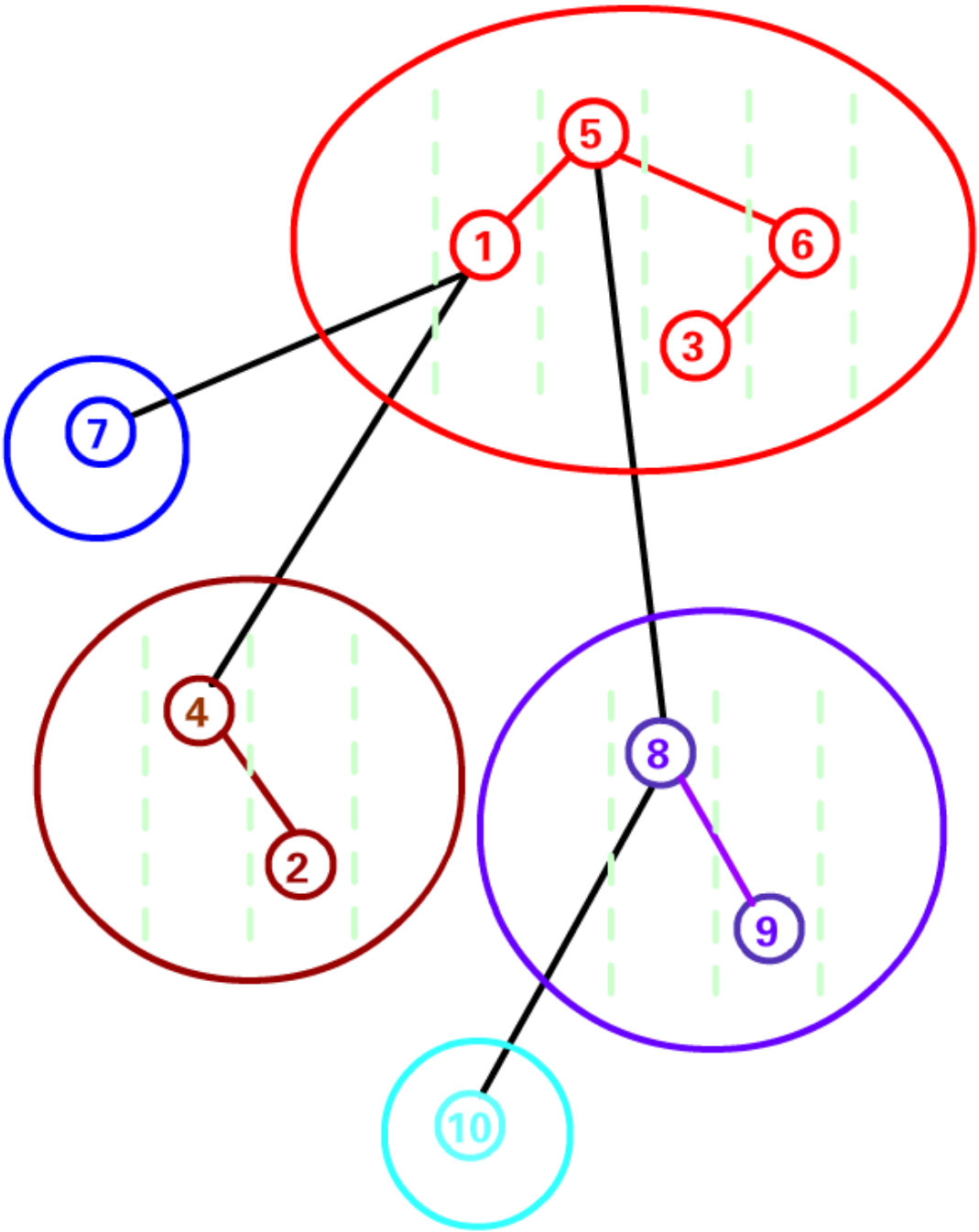
splay(6)然后把 6 的右孩子改为 0

然而并没有什么变化

接下来 splay(5)，把 5 的右孩子改为 6。

变化如下：


1
7
5
4
8
3
2
10
9
6



这样，我们就模拟完了求 6 和 9 的最近公共祖先的全过程，5 最终被旋转到了根部。

三、FindRoot（查找树根）注：查找树根指查找原树的树根，并非 link-cut tree 的树根，准确的说是寻找 link-cut tree 中主链 deep 最小的点。

access 操作是 link-cut tree 一切操作的基础。

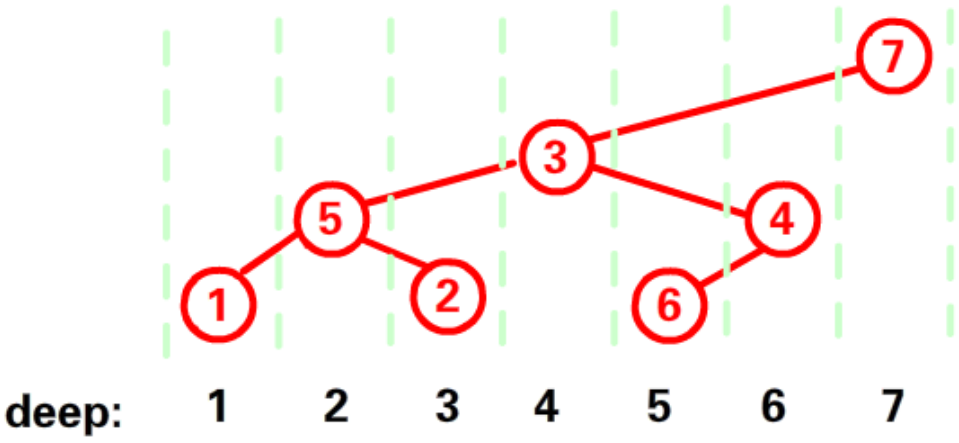
在 ACCESS(v) 之后，根结点一定是 v 所属的 AuxiliaryTree 的最小结点.我们先把 v 旋转到它所属的 AuxiliaryTree 的根.再从 v 开始,沿着 AuxiliaryTree 向左走,直到不能再向左,这个点就是我们要找的根结点.由于使用的是 SplayTree 数据结构保存 AuxiliaryTree，我们还需要对根结点进行 Splay 操作。

这话都好理解…因为在进行完 access 操作之后，主链一定变为从根节点到刚刚访问过的点,在讲解怎么查找 lca 的时候大家一定早就深有体会。

那么我们举个例子模拟一下

如图是一条 1 到 7 的树链。

如果把 7 旋转到根，只需一路需找它的左孩子即可。



我们来写一下代码：

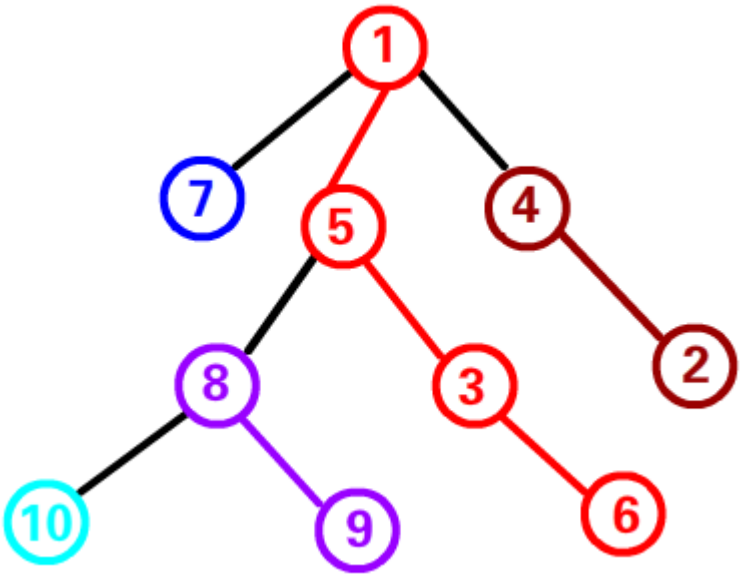
```
int findroot(int root)
{
    access(root);splay(root);int temp=root;
    while(t[temp].ch[0])temp=t[temp].ch[0];
    return temp;
}
```

上述操作我都画出了 **splay tree** 的具体变化，接下来我将不会再管 **splay** 的内部变化，真正把 **splay** 当成一个工具来使用（因为如果你再考虑 **splay** 内部变化的话考虑的东西会很多…让 **splay** 自己考虑自己就行了）我们更需要注意的是宏观的操作，而非 **splay** 内部的微观操作。

四、**makeroot/changeroot**（改变树根）

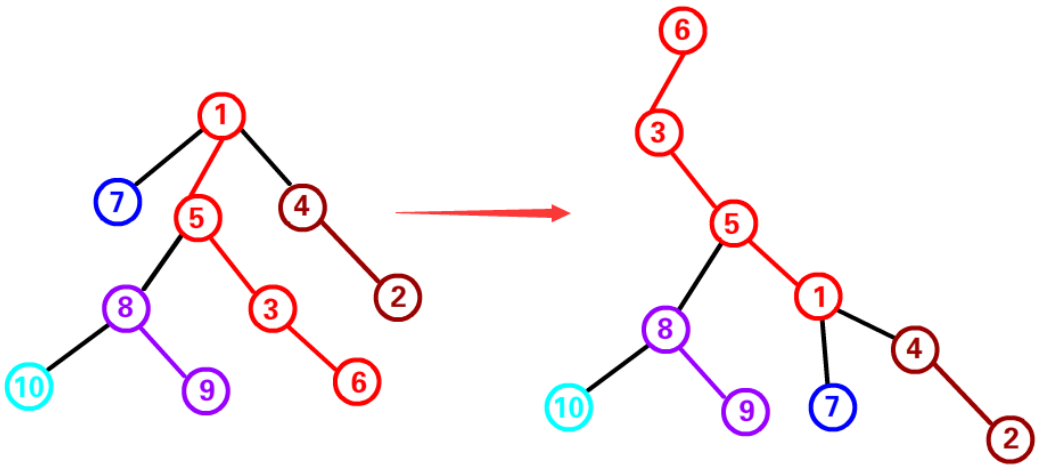
很多时候，我们需要改变原树的树根来进行更多操作，如何改变树根成了一个问题。

我们还用这颗树来举例子吧



首先我们有了 access 操作，如果原来的树根和需要改变成的新树根不在同一条树链上是一个很尴尬的情况…因为我们这样会牵扯到多颗 splay tree,会导致问题变得非常复杂，所以我们首先用 access 操作把原树根和新树根链接到同一条树链上。

比如上图中的树，现在树根是 1，我们把树根换成 6，看看会发生什么?

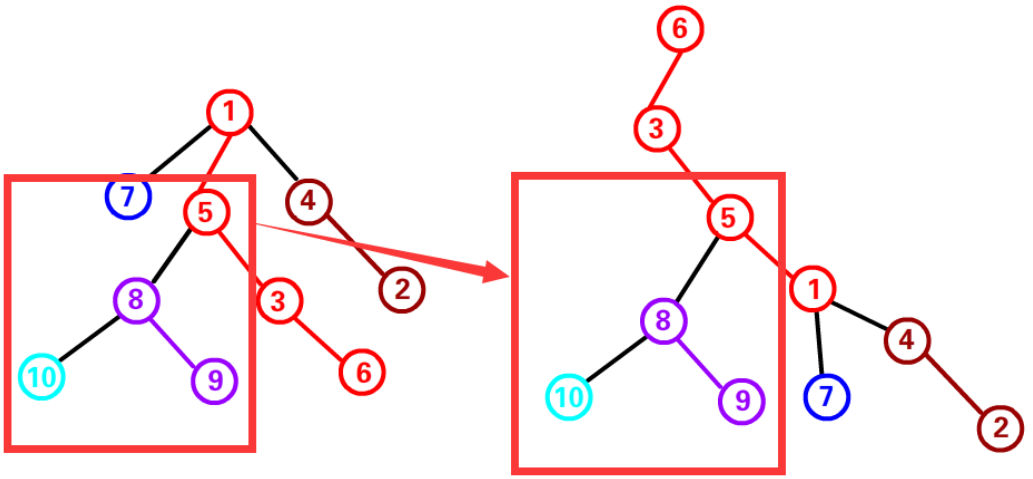


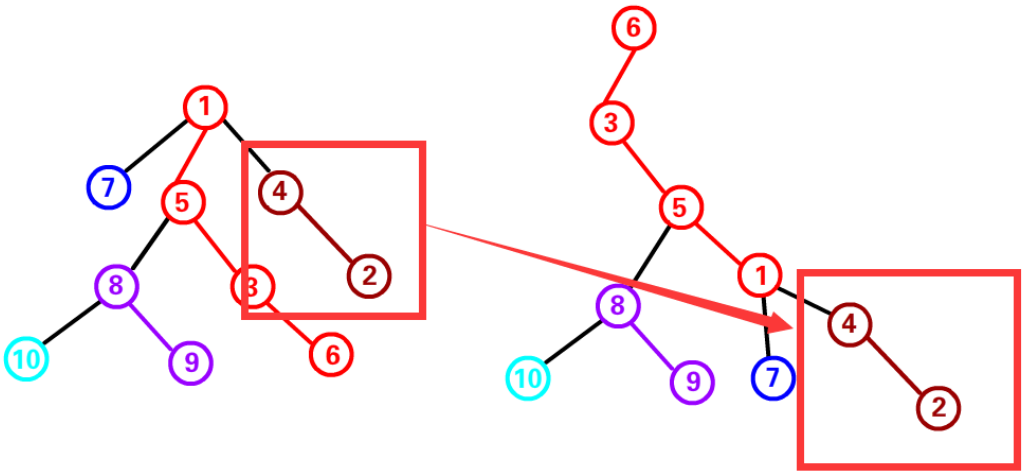
发现什么了么?

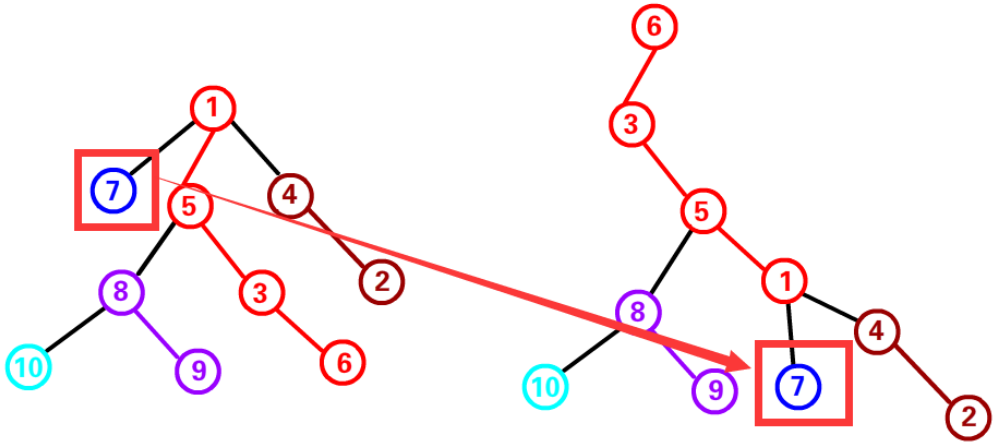
没发现，没关系，我们一起来看看

首先是其他树链，可以看到，因为我们用 access 操作把变化的点都链接到同一条树链了，所以对其他树链都无影响。

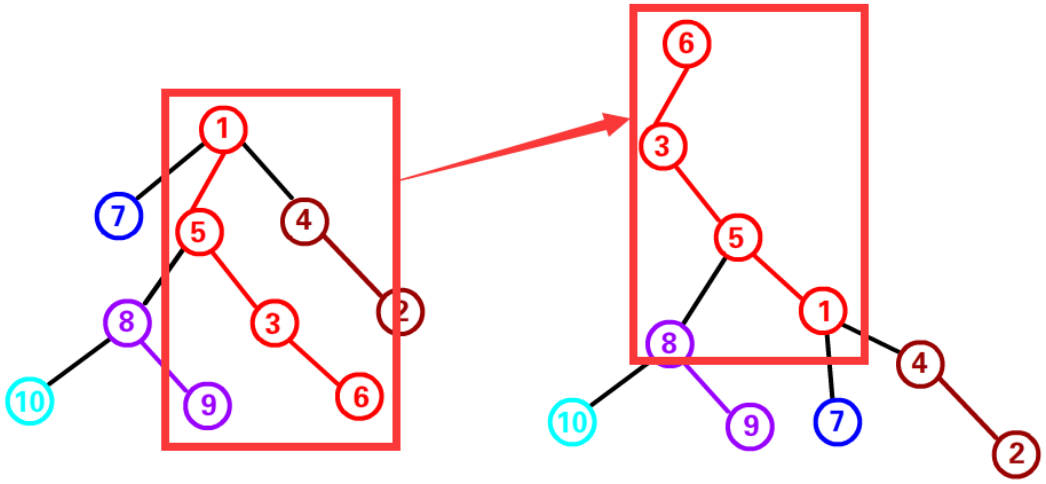
各其他树链如下图







那么主链呢？，我们来观察一下主链



我们可以看到，主链整个翻转过来了，**splay** 的翻转（忘记的话再去写一遍 **splay** 区间翻转吧~）

翻转操作直接打懒标赖着不动即可（同区间翻转）。（具体什么时候释放我们最后与 **splay** 一起说）

具体代码实现

```
void changeroot(int root)
{
        access(root);splay(root);t[root].lazy^=1;
}
```

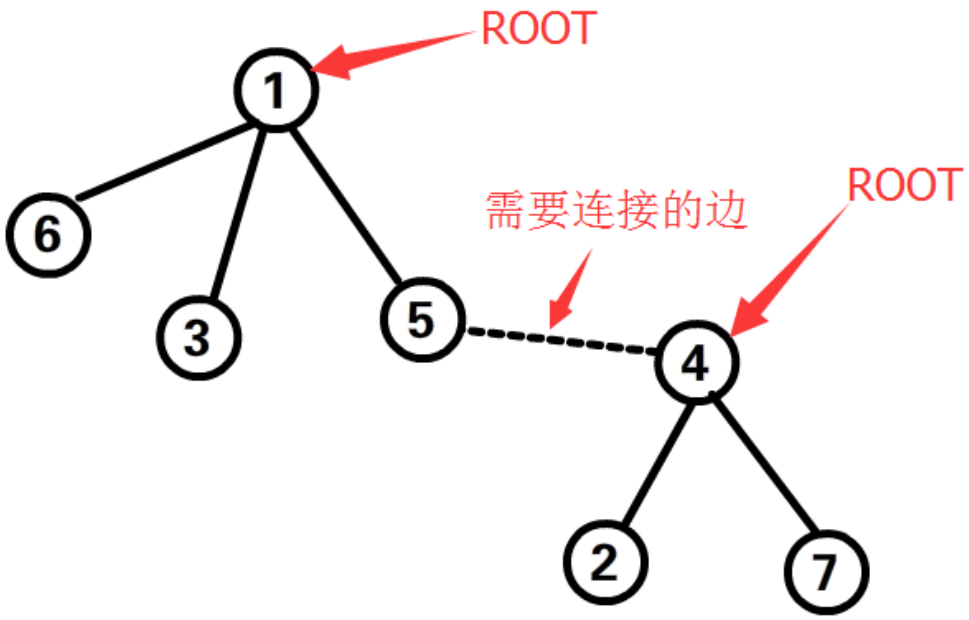
五、join/link（合并两颗 link-cut tree）

先访问 v，然后修改 v 所属的 Auxiliary Tree 的 PathParent 为 w，然后再次访问 v。

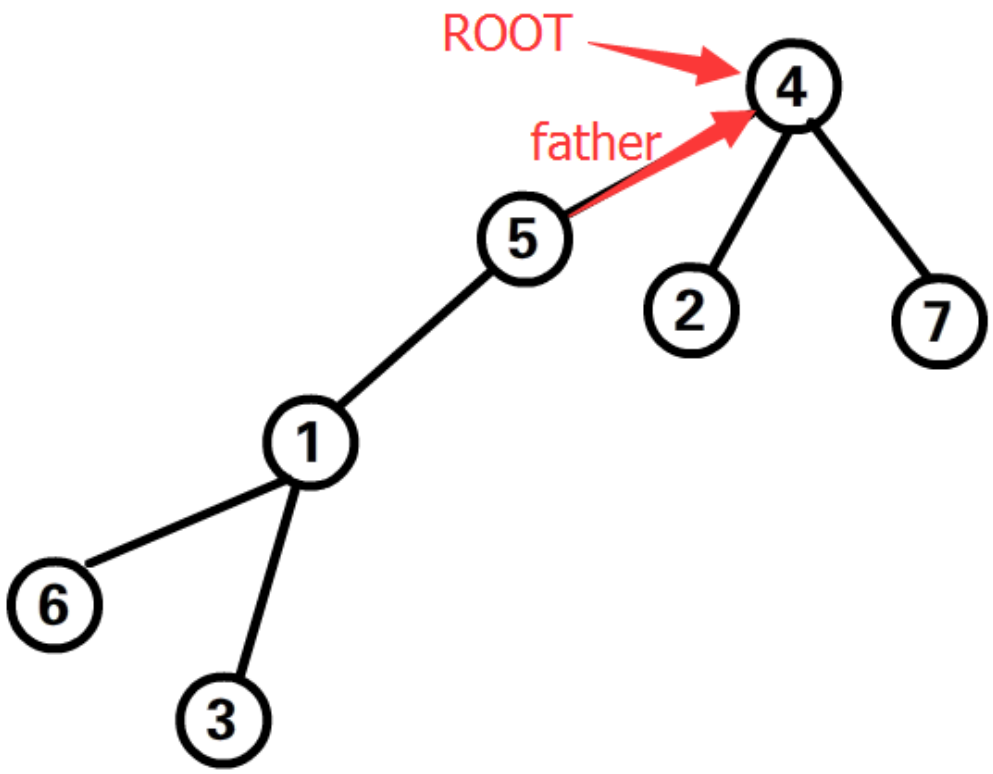
说的步骤有点繁琐…实际上我们并不这么干…我们节省一些步骤，首先把 root1 变为根，然后直接把它的父亲变为 root2 即可。

我们跑一遍吧…

比如现在又两颗树，一颗树的根节点是 1，另一颗树的根节点是 4，我们需要在 4 与 5 之间连一条边。



把第一棵树的根变为 5。，把 5 的父亲指向 4



这里有个很重要的问题…合并完了谁是树根?

树根是 **root2** 所在树的树根…也就是说，实际上当我们多次运行 **link** 之后树根是谁就很难确定了，但是我们并不用担心…因为有 **changeroot** 操作，大不了从新换树根。

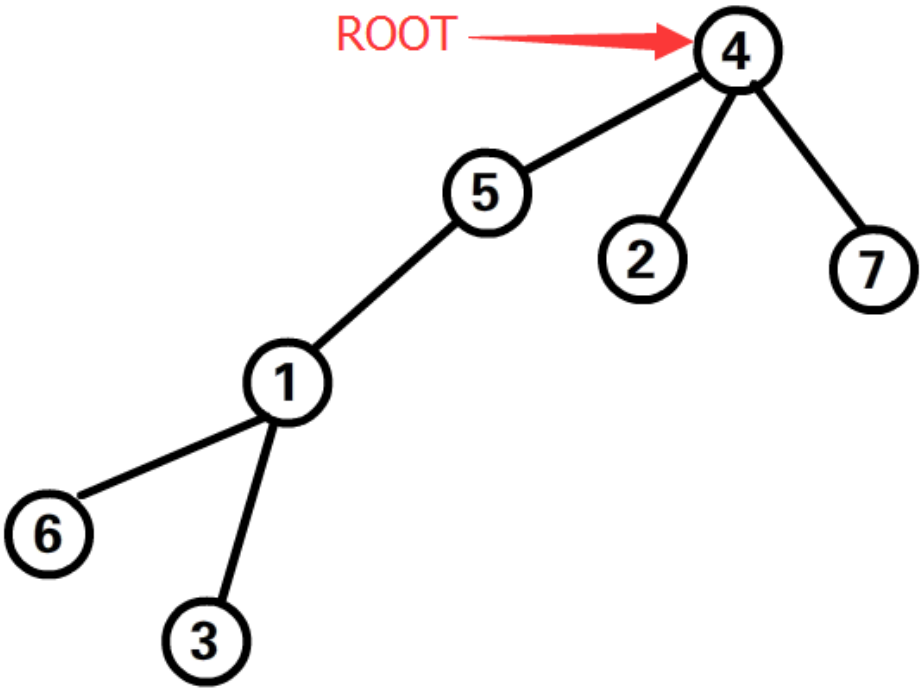
代码其实很好写

```
void link(int root1,int root2)
{
    makeroot(root1);t[root1].fa=root2;
}
```

六、**cut**（拆分）

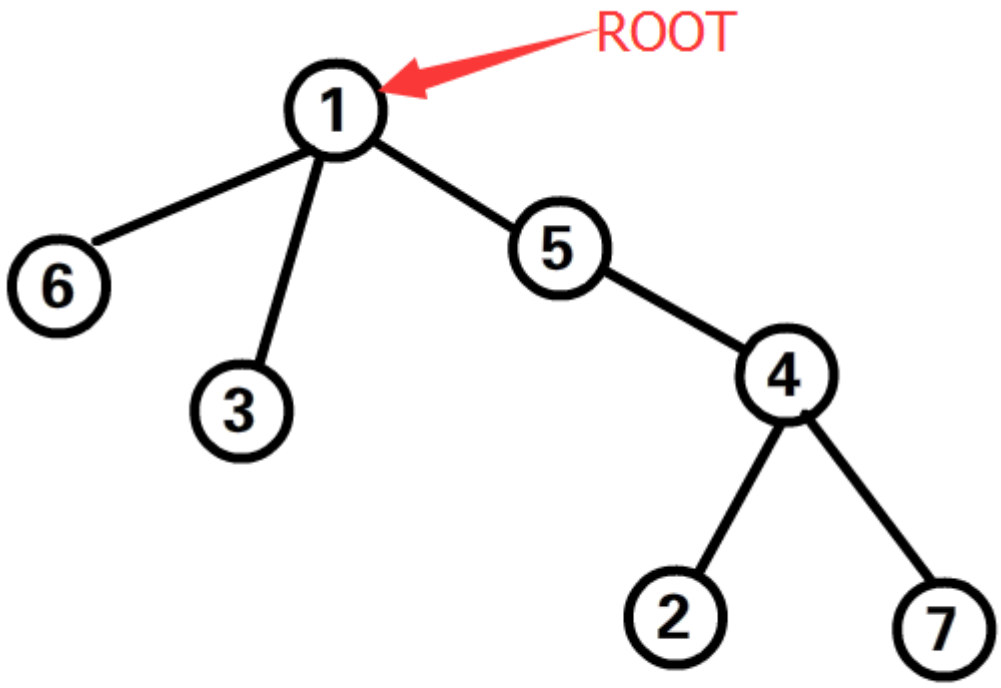
如果 **root1** 和 **root2** 两点在树中是链接好的，那么把

root1 变为树根，去访问 root2，主链一定变为只有 root1 和 root2 两点的一条链，这正是我们想要的，这时再 splay(root2)，root1 一定是 root2 的左孩子。

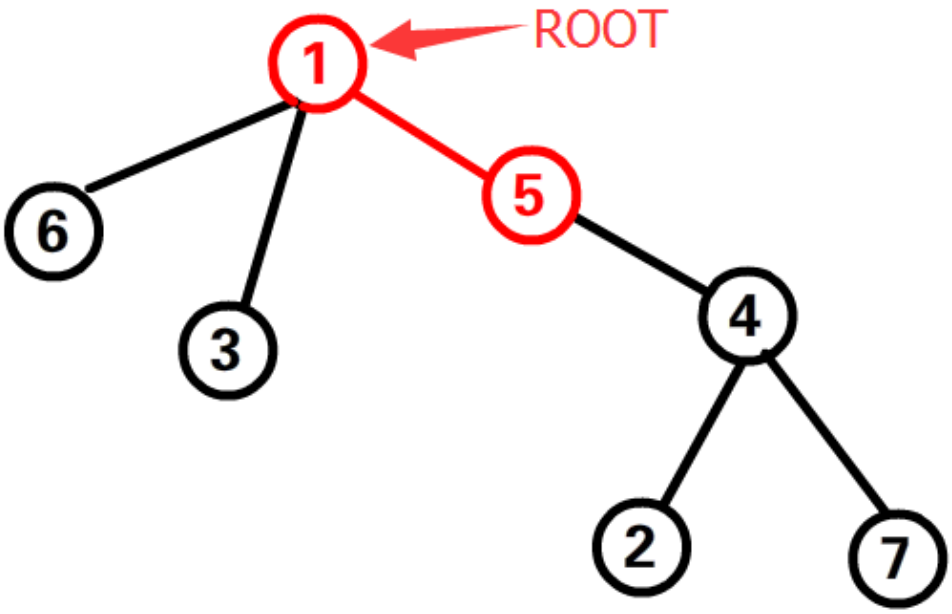


比如现在要断开 1 和 5。

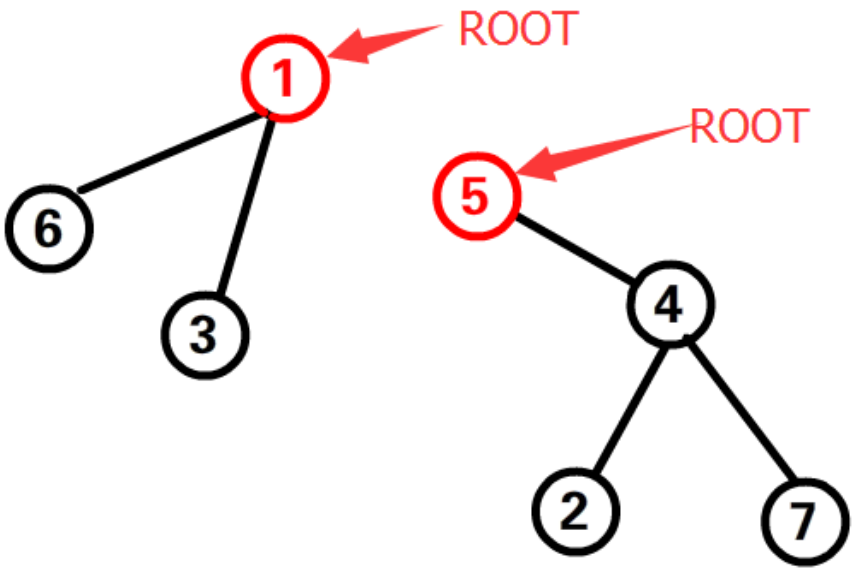
先把 1 变成树根。



构建 1 到 5 的树链。



断开这条树链。



代码：

```
void cut(int root1,int root2)
{
    makeroot(root1);access(root2);splay(root2);
    t[root2].ch[0]=t[root1].fa=0;
}
```

三、link-cut tree 中 splay tree 的具体写法

link-cut tree 中的 **splay** 的写法和平时咱们写的还是有点小区别的。

isroot 函数

判断一个点是否为 **splay tree** 的根节点。我们写 **splay** 时都是看一个点的父节点是否为 **0**，如果这个点的父节点为 **0**，则说明这个点是根节点。但是，因为我们把多棵 **splay** 用 **father** 给连起来了…一个 **splay tree** 的根节点的父节点连向了另一颗 **splay**…怎么办，也好说，因为这颗 **splay** 不会用 **child** 指针连回来了。

代码：

```
bool isroot(int root)
{
    return
t[t[root].fa].ch[0]!=root&&t[t[root].fa].ch[1]!=root;
}
```

旋转

旋转时需要注意，这个点的爷爷节点是不是和自己在同一棵 **splay** 上，如果不在的话，不要把爷爷节点的孩子变成自己，只要把自己的父节点变成爷爷节点即可。

（实际上就多了一句话）

代码：

```
void rot(int root)
{
    int fa=t[root].fa;
    int gfa=t[fa].fa;
    int t1=(root!=t[fa].ch[0]);
    int t2=(fa!=t[gfa].ch[0]);
    int ch=t[root].ch[1^t1];

    if(!isroot(fa))t[gfa].ch[0^t2]=root;          //判断一下父节点
```

是不是根即可。

```
    t[ch].fa=fa;
    t[root].fa=gfa;
    t[root].ch[1^t1]=fa;
    t[fa].fa=root;
    t[fa].ch[0^t1]=ch;                          //如果你需要维护东西的话，
```

在这里 update

```
    return;
}
```

splay

splay 操作时，要注意什么呢，首先，不要 **splay** 到其他树上去，其次…懒标记的释放，我们要一路向上找父节点，然后把父节点到自己路径上所有的点的懒标记全部释放。用到一个小堆栈。

代码：

```
void pushdown(int root)
{
    if(t[root].lazy)
    {
                t[root].lazy=0;
                swap(t[root].ch[0],t[root].ch[1]);
                t[t[root].ch[0]].lazy^=1;
                t[t[root].ch[1]].lazy^=1;
    }
    return;
}
void splay(int root)
{
    stk[++top]=root;
    for(int i=root;!isroot(i);i=t[i].fa)
    {
           stk[++top]=t[i].fa;
    }
    while(top)
```

```
    {
            pushdown(stk[top--]);
    }
    while(!isroot(root))
    {
           int fa=t[root].fa;
           int gfa=t[fa].fa;
           if(!isroot(fa))

root==t[fa].ch[0]^fa==t[gfa].ch[0]?rot(root):rot(fa);
           rot(root);
    }
    return;
}
```

四、link-cut tree 的实战练习

一、动态维护树结构。

洞穴勘测 codevs1839

时间限制: 1 s

空间限制: 256000 KB

题目描述 Description

辉辉热衷于洞穴勘测。某天，他按照地图来到了一片被标记为 JSZX 的洞穴群地区。经过初步勘测，辉辉发现这片区域由 n 个洞穴（分别编号为 1 到 n）以及若干通道组成，并且每条通道连接了恰好两个洞穴。假如两个洞穴可以通过一条或者多条通道按一定顺序连接起来，那么这两个洞穴就是连通的，按顺序连接在一起的这些通道则被称之为这两个洞穴之间的一条路径。

洞穴都十分坚固无法破坏，然而通道不太稳定，时常因为外界影响而发生改变，比如，根据有关仪器的监测结果，123 号洞穴和 127 号洞穴之间有时会出现一条通道，有时

这条通道又会因为某种稀奇古怪的原因被毁。辉辉有一台监测仪器可以实时将通道的每一次改变状况在辉辉手边的终端机上显示:

如果监测到洞穴 u 和洞穴 v 之间出现了一条通道，终端机上会显示一条指令 Connect u v

如果监测到洞穴 u 和洞穴 v 之间的通道被毁，终端机上会显示一条指令 Destroy u v

经过长期的艰苦卓绝的手工推算，辉辉发现一个奇怪的现象：无论通道怎么改变，任意时刻任意两个洞穴之间至多只有一条路径。因而，辉辉坚信这是由于某种本质规律的支配导致的。因而，辉辉更加夜以继日地坚守在终端机之前，试图通过通道的改变情况来研究这条本质规律。

然而，终于有一天，辉辉在堆积成山的演算纸中崩溃了……他把终端机往地面一砸（终端机也足够坚固无法破坏），转而求助于你，说道："你老兄把这程序写写吧"。

辉辉希望能随时通过终端机发出指令 Query u v，向监测仪询问此时洞穴 u 和洞穴 v 是否连通。现在你要为他编写程序回答每一次询问。

已知在第一条指令显示之前，JSZX 洞穴群中没有任何通道存在。

输入描述 Input Description

第一行为两个正整数 n 和 m，分别表示洞穴的个数和终端机上出现过的指令的个数。

以下 m 行，依次表示终端机上出现的各条指令。每行开头是一个表示指令种类的字符串 s("Connect"、"Destroy"或者"Query",区分大小写),之后有两个整数 u 和 v (1≤u, v≤n 且 u≠v) 分别表示两个洞穴的编号。

输出描述 Output Description

对每个 Query 指令,输出洞穴 u 和洞穴 v 是否互相连通:是输出"Yes",否则输出"No"。（不含双引号）

样例输入 Sample Input

```
200 5
Query 123 127
Connect 123 127
Query 123 127
Destroy 127 123
Query 123 127
```

样例输出 Sample Output

```
No
Yes
No
```

数据范围及提示 Data Size & Hint

10%的数据满足 n≤1000, m≤20000

20%的数据满足 n≤2000, m≤40000

30%的数据满足 n≤3000, m≤60000

40%的数据满足 n≤4000, m≤80000

50%的数据满足 n≤5000, m≤100000

60%的数据满足 n≤6000, m≤120000

70%的数据满足 n≤7000, m≤140000

80%的数据满足 n≤8000, m≤160000

90%的数据满足 n≤9000, m≤180000

100%的数据满足 n≤10000, m≤200000

保证所有 Destroy 指令将摧毁的是一条存在的通道

本题输入、输出规模比较大，建议 c\c++选手使用 scanf 和 printf 进行 I\O 操作以免超时

乍一看要用并查集维护，但是发现有删除操作…所以并查集

做不了，然后题目告诉你是棵树，而且总是保证这颗树合法，于是就用 **link-cut trees** 呗。

重点是，怎么判断两点是否联通，这点可以用并查集的思想，看看他们两个是否有相同的根，如果他们两个的有相同的根节点，那么它们就是联通的。

代码：

```
#include<iostream>
#include<cstdlib>
#include<cstring>
#include<cstdio>
#include<algorithm>
using namespace std;
const int MAXN=100005;
int n,m,a1,b1;
char q[10];
struct tree
{
      int fa,ch[2],lazy;
}t[MAXN];
int stk[MAXN],top;
void pushdown(int root)
{
    if(t[root].lazy)
    {
                t[root].lazy=0;
                swap(t[root].ch[0],t[root].ch[1]);
                t[t[root].ch[0]].lazy^=1;
                t[t[root].ch[1]].lazy^=1;
    }
    return;
}
bool isroot(int root)
{
    return
t[t[root].fa].ch[0]!=root&&t[t[root].fa].ch[1]!=root;
}
void rot(int root)
{
```

```
    int fa=t[root].fa;
    int gfa=t[fa].fa;
    int t1=(root!=t[fa].ch[0]);
    int t2=(fa!=t[gfa].ch[0]);
    int ch=t[root].ch[1^t1];
    if(!isroot(fa))t[gfa].ch[0^t2]=root;
    t[ch].fa=fa;
    t[root].fa=gfa;
    t[root].ch[1^t1]=fa;
    t[fa].fa=root;
    t[fa].ch[0^t1]=ch;
    return;
}
void splay(int root)
{
    stk[++top]=root;
    for(int i=root;!isroot(i);i=t[i].fa)
    {
          stk[++top]=t[i].fa;
    }
    while(top)
    {
           pushdown(stk[top--]);
    }
    while(!isroot(root))
    {
          int fa=t[root].fa;
          int gfa=t[fa].fa;
          if(!isroot(fa))

root==t[fa].ch[0]^fa==t[gfa].ch[0]?rot(root):rot(fa);
          rot(root);
    }
    return;
}
void access(int root)
{
    int temp=0;
    while(root)
    {
             splay(root);
             t[root].ch[1]=temp;
             temp=root;
             root=t[root].fa;
```

```
    }
    return;
}
void makeroot(int root)
{
    access(root);splay(root);t[root].lazy^=1;
}
void link(int root1,int root2)
{
    makeroot(root1);t[root1].fa=root2;
}
void cut(int root1,int root2)
{
    makeroot(root1);access(root2);splay(root2);
    t[root2].ch[0]=t[root1].fa=0;
}
int findroot(int root)
{
   access(root);splay(root);int temp=root;
   while(t[temp].ch[0])temp=t[temp].ch[0];
   return temp;
}
int main()
{
   scanf("%d %d",&n,&m);
   for(int i=1;i<=m;++i)
   {
          scanf("%s %d %d",&q,&a1,&b1);
          if(q[0]=='Q')
          {
                   if(findroot(a1)==findroot(b1))
                   {
                        printf("Yes\n");
                   }
                   else
                   {
                      printf("No\n");
                   }
          }
          if(q[0]=='C')
          {
                     link(a1,b1);
          }
          if(q[0]=='D')
```

```
            {
                        cut(a1,b1);
            }
    }
    return 0;
}
```

二、动态树维护简单区间问题

弹飞绵羊 codevs2333

2010 年省队选拔赛湖南

时间限制：1 s

空间限制：128000 KB

题目等级 ：大师 Master

题解

题目描述 Description

Lostmonkey 发明了一种超级反弹装置。为了在绵羊朋友面前显摆，他邀请小绵羊一起玩个游戏。游戏一开始，Lostmonkey 在地上沿一条直线摆放 n 个反弹装置，并按从前往后的方式将反弹装置依次编号为 0 到 n-1，对 0≤i≤n-1，为第 i 个反弹装置设定了初始弹力系数 ki，当绵羊落到第 i 个反弹装置上时，它将被往后弹出 ki 步，即落到第 i+ki 个反弹装置上，若不存在第 i+ki 个反弹装置，则绵羊被弹飞。绵羊想知道： 从第 i 个反弹装置开始， 它被弹出几次 （含被弹飞的那次）后会被弹飞。为使游戏更有趣，Lostmonkey 还可以修改某个反弹装置的弹力系数，但任何时候弹力系数均为正整数。

输入描述 Input Description

输入文件第一行是一个整数 n，表示地上摆放 n 个反弹装置，输入文件第二行是用空格隔开的 n 个正整数 k0,k1,…,kn-1，分别表示 n 个反弹装置的初始弹力系数。输入文件第三行是一个正整数 m，表示后面还有 m 行输入数据。接下来的 m 行，每行至少有用空格隔开的两个整数 i 和 j，若 i=1，则你要输出从第 j 个反弹装置开始，被弹出几次后会被弹飞；若 i=2，则该行有用空格隔开的三个整数 i，j 和 k，表示第 j 个反弹装置的弹力系数被修改为 k。

输出描述 Output Description

包含的行数等于输入文件最后 m 行中 i=1 的行数。第 h 行输出一个整数，表示对输入中给出的第 h 个求弹出次数的问题，基于 n 个反弹装置当时的弹力系数，求出的弹出次数。

样例输入 Sample Input

```
4
1 2 1 1
3
1 1
2 1 1
1 1
```

样例输出 Sample Output

```
2
3
```

数据范围及提示 Data Size & Hint

输入的数据保证 20%的数据满足 n,m≤10000。100%的数据满足 n≤200000,m≤100000

这个题号和这个题目….2333333333 codevs 你故意的

首先这个题是一个动态图，那么它是不是动态树？如果把图画出来，会发现其实这个题是个森林，如果把绵羊弹飞的点

作为第 **n+1** 个点，这道题就是从某点到点 **n+1** 经过的点数和**-1**。

那么，这道题有什么特殊的地方么？没有**=-=**

我只是借这道题来说一下 **update** 函数在更新时的小优化，以及如果出现了问题如何用 **debug** 函数调试（如果写出一个复杂的东西出现一点小毛病…真是酸爽）

我们来看代码吧

```
#include<iostream>
#include<cstdlib>
#include<cstdio>
#include<cstring>
#include<algorithm>
using namespace std;
const int MAXN=200005;
int stk[MAXN],top,next[MAXN],n,m,que,a1,b1;
struct tree
{
   int fa,ch[2],size,lazy;
}t[MAXN];
int nexts(int x,int y)
{
   if(x+y>n)return n+1;
   return x+y;
}
void updata(int root)
{
   t[root].size=t[t[root].ch[0]].size+t[t[root].ch[1]].size+1;
   return;
}
void pushdown(int root)
{
   if(t[root].lazy)
   {
      swap(t[root].ch[0],t[root].ch[1]);
      t[root].lazy=0;
      t[t[root].ch[0]].lazy^=1;
      t[t[root].ch[1]].lazy^=1;
```

```
    }
    return;
}
bool isroot(int root)
{
    return root!=t[t[root].fa].ch[0]&&root!=t[t[root].fa].ch[1];
}
```

void debug() //这个就是 debug 函数…没有错，就是从 0 到 n 的所有信息都输出一遍。

```
{
    for(int i=0;i<=n+1;++i)
    {
        cout<<i<<" "<<t[i].fa<<" "<<t[i].ch[0]<<" "<<t[i].ch[1]<<"
"<<t[i].size<<" "<<t[i].lazy<<endl;
    }
}
void rot(int root)
{
    int fa=t[root].fa;
    int gfa=t[fa].fa;
    int t1=(root!=t[fa].ch[0]);
    int t2=(fa!=t[gfa].ch[0]);
    int ch=t[root].ch[1^t1];
    t[ch].fa=fa;
    if(!isroot(fa))t[gfa].ch[0^t2]=root;
    t[root].fa=gfa;
    t[root].ch[1^t1]=fa;
    t[fa].fa=root;
    t[fa].ch[0^t1]=ch;
    updata(fa);                    //是不是发现少了一句话？少了
update(root);
    return;
}
void splay(int root)
{
    stk[++top]=root;
    for(int i=root;!isroot(i);i=t[i].fa)
    {
        stk[++top]=t[i].fa;
    }
    while(top)
```

```
    {
        pushdown(stk[top--]);
    }
    while(!isroot(root))
    {
        int fa=t[root].fa;
        int gfa=t[fa].fa;
        if(!isroot(fa))
        {
            root==t[fa].ch[0]^fa==t[gfa].ch[0]?rot(root):rot(fa);
        }
        rot(root);
    }
    updata(root);                    //因为 rot 函数少了一句,这里一定要补回
来
    return;
}
void access(int root)
{
    int temp=0;
    while(root)
    {
        splay(root);
        t[root].ch[1]=temp;
        updata(root);
        temp=root;
        root=t[root].fa;
    }
    return;
}
void makeroot(int root)
{
    access(root);splay(root);t[root].lazy^=1;
    return;
}
void link(int root1,int root2)
{
    makeroot(root1);t[root1].fa=root2;
    return;
}
void cut(int root1,int root2)
{
```

```
    makeroot(root1);access(root2);splay(root2);
    if(t[root2].ch[0]==root1)t[root2].ch[0]=t[root1].fa=0;
    return;
}
void work(int root1,int root2)
{
    makeroot(root1);access(root2);splay(root2);
    return;
}
int main()
{
    scanf("%d",&n);
    for(int i=1;i<=n+1;++i)
    {
        t[i].size=1;
    }
    for(int i=1;i<=n;++i)
    {
        scanf("%d",&a1);
        next[i]=nexts(i,a1);
        link(i,next[i]);
    }
    scanf("%d",&m);
    for(int i=1;i<=m;++i)
    {
        scanf("%d",&que);
        if(que==1)
        {
            scanf("%d",&a1);
            ++a1;
            work(a1,n+1);
            printf("%d\n",t[n+1].size-1);
        }
        if(que==2)
        {
            scanf("%d %d",&a1,&b1);
            ++a1;
            int temp=nexts(a1,b1);
            if(temp!=next[a1])
            {
                cut(a1,next[a1]);
                next[a1]=temp;
                link(a1,next[a1]);
            }
```

```
        }
    }
    return 0;
}
```

减少几次 **update** 会快很多么？嗯…这道题并不会，但是有的题使用 **splay** 维护的信息相当的多，少使用几次 **update** 会有奇效。

常见错误和解决方法：

1、 死循环，多半是 **splay** 写错了，在 **splay** 中每次输出 **root** 和父节点以及爷爷节点。

2、 结果不正确，这时需要在 **work** 函数之后使用 **debug**，首先看 **root1** 是不是根节点，其次是从 **root2** 到 **root1** 是否构成主链（检查树的结构是否正确），如果不正确则检查 **access** 函数。如果树结构正确，检查从 **root1** 到 **root2** 的懒标记是否全部释放以及子树懒标记是否会影响到经过的节点。否则检查 **update** 函数。

三、边权？点权！

小机房的树 codevs 2370

时间限制：1 s

空间限制：256000 KB

题目等级 ：钻石 Diamond

题解

题目描述 Description

小机房有棵焕狗种的树，树上有 N 个节点，节点标号为 0 到 N-1，有两只虫子名叫飘狗

和大吉狗，分居在两个不同的节点上。有一天，他们想爬到一个节点上去搞基，但是作为两只虫子，他们不想花费太多精力。已知从某个节点爬到其父亲节点要花费 c 的能量（从父亲节点爬到此节点也相同），他们想找出一条花费精力最短的路，以使得搞基的时候精力旺盛，他们找到你要你设计一个程序来找到这条路，要求你告诉他们最少需要花费多少精力

输入描述 Input Description

第一行一个 n，接下来 n-1 行每一行有三个整数 u，v，c 。表示节点 u 爬到节点 v 需要花费 c 的精力。

第 n+1 行有一个整数 m 表示有 m 次询问。接下来 m 行每一行有两个整数 u ，v 表示两只虫子所在的节点

输出描述 Output Description

一共有 m 行，每一行一个整数，表示对于该次询问所得出的最短距离。

样例输入 Sample Input

```
3
1 0 1
2 0 1
3
1 0
2 0
1 2
```

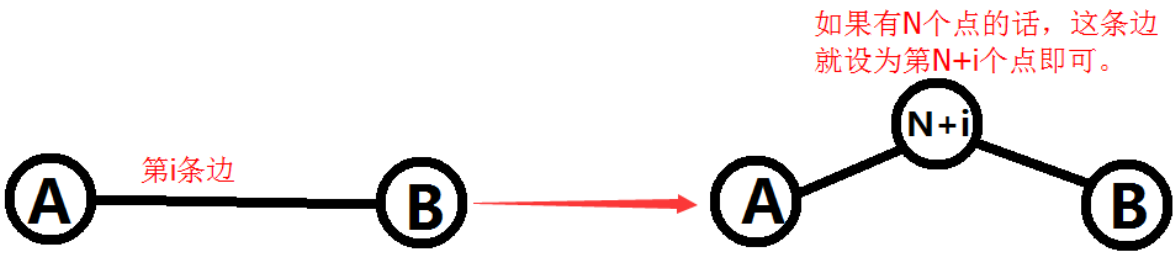
样例输出 Sample Output

```
1
1
2
```

数据范围及提示 Data Size & Hint

1<=n<=50000， 1<=m<=75000， 0<=c<=1000

一道比较裸的 **lca** 题，但是我们用 **link-cut tree** 来做一下，可能你已经注意到了，**link-cut tree** 无法维护边的信息。



代码：

```
#include<iostream>
#include<cstdlib>
#include<cstdio>
#include<cstring>
#include<algorithm>
using namespace std;
const int MAXN=225005;
int stk[MAXN],top,n,m,a1,b1,c1;
struct tree
{
        int fa,ch[2],num,sum,lazy;
}t[MAXN];
void pushdown(int root)
{
        if(t[root].lazy)
        {
                t[root].lazy=0;
                swap(t[root].ch[0],t[root].ch[1]);
                t[t[root].ch[0]].lazy^=1;
                t[t[root].ch[1]].lazy^=1;
        }
        return;
}
void updata(int root)
```

```
{

t[root].sum=t[t[root].ch[0]].sum+t[t[root].ch[1]].sum+t[root].
num;
        return;
}
bool isroot(int root)
{
        return
root!=t[t[root].fa].ch[0]&&root!=t[t[root].fa].ch[1];
}
void rot(int root)
{
        int fa=t[root].fa;
        int gfa=t[fa].fa;
        int t1=(root!=t[fa].ch[0]);
        int t2=(fa!=t[gfa].ch[0]);
        int ch=t[root].ch[1^t1];
        if(!isroot(fa))t[gfa].ch[0^t2]=root;
        t[ch].fa=fa;
        t[root].fa=gfa;
        t[root].ch[1^t1]=fa;
        t[fa].fa=root;
        t[fa].ch[0^t1]=ch;
        updata(fa);
        return;
}
void splay(int root)
{
        stk[++top]=root;
        for(int i=root;!isroot(i);i=t[i].fa)
        {
                stk[++top]=t[i].fa;
        }
        while(top)
        {
                pushdown(stk[top--]);
        }
        while(!isroot(root))
        {

                int fa=t[root].fa;
                int gfa=t[fa].fa;
                if(!isroot(fa))
```

```
                {
(root==t[fa].ch[0])^(fa==t[gfa].ch[0])?rot(root):rot(fa);
                }
                rot(root);
        }
        updata(root);
        return;
}
void access(int root)
{
        int temp=0;
        while(root)
        {
                splay(root);
                t[root].ch[1]=temp;
                updata(root);
                temp=root;
                root=t[root].fa;
        }
}
void changeroot(int root)
{
        access(root);splay(root);t[root].lazy^=1;
}
void link(int root1,int root2)
{
        changeroot(root1);t[root1].fa=root2;
}
void work(int root1,int root2)
{
        changeroot(root1);access(root2);splay(root2);
}
int main()
{
        scanf("%d",&n);
        for(int i=1;i<n;++i)
        {
                scanf("%d %d %d",&a1,&b1,& t[n+i].num);
                ++a1;
                ++b1;
                t[n+i].sum=t[n+i].num;
                link(a1,n+i);
                link(n+i,b1);
```

```
        }
        scanf("%d",&m);
        for(int i=1;i<=m;++i)
        {
                scanf("%d %d",&a1,&b1);
                ++a1;
                ++b1;
                work(a1,b1);
                printf("%d\n",t[b1].sum);
        }
        return 0;
}
```

四、静态树维护较复杂区间问题

树的统计 2008 年省队选拔赛浙江 codevs 2460

时间限制: 2 s

空间限制: 128000 KB

题目等级 : 大师 Master

题解

题目描述 Description

一棵树上有 n 个节点，编号分别为 1 到 n，每个节点都有一个权值 w。

我们将以下面的形式来要求你对这棵树完成一些操作:

I. CHANGE u t : 把结点 u 的权值改为 t

II. QMAX u v: 询问从点 u 到点 v 的路径上的节点的最大权值

III. QSUM u v: 询问从点 u 到点 v 的路径上的节点的权值和

注意: 从点 u 到点 v 的路径上的节点包括 u 和 v 本身

输入描述 Input Description

输入文件的第一行为一个整数 n，表示节点的个数。

接下来 n - 1 行，每行 2 个整数 a 和 b，表示节点 a 和节点 b 之间有一条边相连。

接下来 n 行，每行一个整数，第 i 行的整数 wi 表示节点 i 的权值。

接下来 1 行，为一个整数 q，表示操作的总数。

接下来 q 行，每行一个操作，以“CHANGE u t”或者“QMAX u v”或者“QSUM u v”的形式给出。

输出描述 Output Description

对于每个“QMAX”或者“QSUM”的操作,每行输出一个整数表示要求输出的结果。

样例输入 Sample Input

```
4
1 2
2 3
4 1
4 2 1 3
12
QMAX 3 4
QMAX 3 3
QMAX 3 2
QMAX 2 3
QSUM 3 4
QSUM 2 1
CHANGE 1 5
QMAX 3 4
CHANGE 3 6
QMAX 3 4
QMAX 2 4
QSUM 3 4
```

样例输出 Sample Output

```
4
1
2
2
10
6
```

5
6
5
16

数据范围及提示 Data Size & Hint

对于 100%的数据，保证 1<=n<=30000，0<=q<=200000；中途操作中保证每个节点的权值 w 在-30000 到 30000 之间。

其实只用说明一个问题，如何修改单点?

把它 **splay** 到树根，直接修改即可。

代码:

```
#include<iostream>
#include<cstdlib>
#include<cstdio>
#include<cstring>
#include<algorithm>
using namespace std;
const int MAXN=30005;
int stk[MAXN],top,n,m,a1,b1;
char q[55];
struct tree
{
   int ch[2],num,fa,lazy,maxx,sums;
}t[MAXN];
struct save_edge
{
   int from,to;
}e[MAXN];
int max1(int x,int y)
{
   return x>y?x:y;
}
bool isroot(int root)
{
   return root!=t[t[root].fa].ch[0]&&root!=t[t[root].fa].ch[1];
}
void updata(int root)
{
   t[root].maxx=max1(t[root].num,max1(t[t[root].ch[0]].maxx,t[
```

```
t[root].ch[1]].maxx));
    t[root].sums=t[t[root].ch[0]].sums+t[t[root].ch[1]].sums+t[
root].num;
    return;
}
void pushdown(int root)
{
    if(t[root].lazy)
    {
        swap(t[root].ch[0],t[root].ch[1]);
        t[root].lazy=0;
        t[t[root].ch[0]].lazy^=1;
        t[t[root].ch[1]].lazy^=1;
    }
    return;
}
void rot(int root)
{
    int fa=t[root].fa;
    int gfa=t[fa].fa;
    int t1=(root!=t[fa].ch[0]);
    int t2=(fa!=t[gfa].ch[0]);
    int ch=t[root].ch[1^t1];
    t[ch].fa=fa;
    if(!isroot(fa))t[gfa].ch[0^t2]=root;
    t[root].fa=gfa;
    t[root].ch[1^t1]=fa;
    t[fa].ch[0^t1]=ch;
    t[fa].fa=root;
    updata(fa);
    return;
}
void splay(int root)
{
    stk[++top]=root;
    for(int i=root;!isroot(i);i=t[i].fa)
    {
        stk[++top]=t[i].fa;
    }
    while(top)
    {
        pushdown(stk[top--]);
    }
    int &fa=t[root].fa;
```

```
    while(!isroot(root))
    {
        int gfa=t[fa].fa;
        if(!isroot(fa))
        {
            root==t[fa].ch[0]^fa==t[gfa].ch[0]?rot(root):rot(fa);
        }
        rot(root);
    }
    updata(root);
    return;
}

void access(int root)
{
    int temp=0;
    while(root)
    {
        splay(root);
        t[root].ch[1]=temp;
        updata(root);
        temp=root;
        root=t[root].fa;
    }
}
void makeroot(int root)
{
    access(root);splay(root);t[root].lazy^=1;
}
void link(int root1,int root2)
{
    makeroot(root1);t[root1].fa=root2;
}
void work(int root1,int root2)
{
    makeroot(root1);access(root2);splay(root2);
}
int main()
{
    scanf("%d",&n);
    for(int i=1;i<n;++i)
    {
        scanf("%d %d",&e[i].from,&e[i].to);
    }
```

```
    for(int i=1;i<=n;++i)
    {
        scanf("%d",&t[i].num);
        t[i].maxx=t[i].sums=t[i].num;
    }
    for(int i=1;i<n;++i)
    {
        link(e[i].from,e[i].to);
    }
    t[0].sums=0;t[0].maxx=-2147483647;
    scanf("%d",&m);
    for(int i=1;i<=m;++i)
    {
        scanf("%s %d %d",&q,&a1,&b1);
        if(q[0]=='C')
        {
            splay(a1);
            t[a1].num=b1;
            updata(a1);
        }
        if(q[0]=='Q')
        {
            work(a1,b1);
            if(q[1]=='M')
            {
                printf("%d\n",t[b1].maxx);
            }
            if(q[1]=='S')
            {
                printf("%d\n",t[b1].sums);
            }
        }
    }
    return 0;
}
```

五、更加自如的使用懒标记以及更加复杂的 update

染色 codevs1566

省队选拔赛山东

时间限制：2 s

空间限制：256000 KB

题目等级 ：大师 Master

题解

题目描述 Description

给定一棵有 n 个节点的无根树和 m 个操作，操作有 2 类:

1、将节点 a 到节点 b 路径上所有点都染成颜色 c;

2、询问节点 a 到节点 b 路径上的颜色段数量（连续相同颜色被认为是同一段），如“112221”由 3 段组成：“11”、“222”和“1”。

请你写一个程序依次完成这 m 个操作。

输入描述 Input Description

第一行包含 2 个整数 n 和 m，分别表示节点数和操作数;

第二行包含 n 个正整数表示 n 个节点的初始颜色

下面 n-1 行每行包含两个整数 和 ，表示 x 和 y 之间有一条无向边。

下面 m 行每行描述一个操作:

“C a b c”表示这是一个染色操作，把节点 a 到节点 b 路径上所有点（包括 a 和 b）都染成颜色 c;

“Q a b”表示这是一个询问操作，询问节点 a 到节点 b（包括 a 和 b）路径上的颜色段数量。

输出描述 Output Description

对于每个询问操作，输出一行答案。

样例输入 Sample Input

```
6 5
2 2 1 2 1 1
1 2
1 3
2 4
2 5
2 6
Q 3 5
C 2 1 1
Q 3 5
C 5 1 2
Q 3 5
```

样例输出 Sample Output

```
3
1
2
```

数据范围及提示 Data Size & Hint

对于 100%的数据 $1\le n\le 10^4$，$1\le m\le 10^5$，$1\le m\le 10^9$;

这道题涉及到了区间修改，怎么办？我们打上懒标记即可，但是这样就会出现多个标记并存的问题，我们必须考虑到标记之间是否互相影响，这道题使用到的两个懒标记是旋转标记和修改标记，它们并不互相影响，所以无所谓啦。（举个例子来说，修改标记和增加标记就是相互影响的两个标记）同时，我们发现这道题在某个点求解时如果子树有没有释放的懒标记会导致 **WA**(类似于线段树求极值),所以在 **update** 的时候我们把子树 **pushdown** 即可。如果你发现你求的极小值都为 0，那么你需要把虚点 0 的极小值赋为负无穷。

这道题你基本上不可能一次写成…如果真的一次写成了，说明你已经很熟练的掌握 **link-cut tree**，不然的话可以好

好练习一下调试技巧…这道题我还是比较推荐的。

代码：

```
#include<iostream>
#include<cstdlib>
#include<cstring>
#include<cstdio>
#include<algorithm>
using namespace std;
const int MAXN=100005;
int n,a1,b1,c1,m,stk[MAXN],top;
char q[5];
struct tree
{
      int ch[2],fa,lx,rx,num,col,lazy1,lazy2;
}t[MAXN];
void debug()
{
     for(int i=0;i<=n;++i)
     {
            cout<<i<<"     fa:"<<t[i].fa<<"     lx:"<<t[i].lx<<"
rx:"<<t[i].rx<<"    ch0:"<<t[i].ch[0]<<"    ch1:"<<t[i].ch[1]<<"
num:"<<t[i].num<<"   col:"<<t[i].col<<"   lazy1:"<<t[i].lazy1<<"
lazy2:"<<t[i].lazy2<<endl;
     }
}
void pushdown(int root)
{
     if(t[root].lazy1)
     {
          swap(t[root].ch[0],t[root].ch[1]);
          swap(t[root].lx,t[root].rx);
          t[root].lazy1=0;
          t[t[root].ch[0]].lazy1^=1;
          t[t[root].ch[1]].lazy1^=1;
     }
     if(t[root].lazy2)
     {

t[t[root].ch[1]].lazy2=t[t[root].ch[0]].lazy2=t[root].lx=t[roo
t].rx=t[root].lazy2;
          t[root].num=1;
          t[root].col=t[root].lazy2;
```

```
          t[root].lazy2=0;
     }
     return;
}
void updata(int root)
{
     if(t[root].ch[0])pushdown(t[root].ch[0]);
     if(t[root].ch[1])pushdown(t[root].ch[1]);
     t[root].num=t[t[root].ch[0]].num+t[t[root].ch[1]].num+1;
     if(t[t[root].ch[0]].rx==t[root].col)--t[root].num;
     if(t[t[root].ch[1]].lx==t[root].col)--t[root].num;
     t[root].lx=t[root].ch[0]?t[t[root].ch[0]].lx:t[root].col;
     t[root].rx=t[root].ch[1]?t[t[root].ch[1]].rx:t[root].col;
}
bool isroot(int root)
{
     return
root!=t[t[root].fa].ch[0]&&root!=t[t[root].fa].ch[1];
}
void rot(int root)
{
     int fa=t[root].fa;
     int gfa=t[fa].fa;
     int t1=(root!=t[fa].ch[0]);
     int t2=(fa!=t[gfa].ch[0]);
     int ch=t[root].ch[1^t1];
     if(!isroot(fa))t[gfa].ch[0^t2]=root;
     t[ch].fa=fa;
     t[root].fa=gfa;
     t[root].ch[1^t1]=fa;
     t[fa].fa=root;
     t[fa].ch[0^t1]=ch;
     updata(fa);
     return;
}
void splay(int root)
{
     stk[++top]=root;
     for(int i=root;!isroot(i);i=t[i].fa)
     {
            stk[++top]=t[i].fa;
     }
     while(top)
     {
```

```
            pushdown(stk[top--]);
    }
    while(!isroot(root))
    {
        int fa=t[root].fa;
        int gfa=t[fa].fa;
        if(!isroot(fa))
        {

root==t[fa].ch[0]^fa==t[gfa].ch[0]?rot(root):rot(fa);
        }
        rot(root);
    }
    updata(root);
    return;
}
void access(int root)
{
    int temp=0;
    while(root)
    {
            splay(root);
            t[root].ch[1]=temp;
            updata(root);
            temp=root;
            root=t[root].fa;
    }
    return;
}
void makeroot(int root)
{
    access(root);splay(root);t[root].lazy1^=1;
}
void link(int root1,int root2)
{
    makeroot(root1);t[root1].fa=root2;
}
void change(int root1,int root2,int cols)
{

makeroot(root1);access(root2);splay(root2);t[root2].lazy2=cols
;
}
void work(int root1,int root2)
```

```
{
    makeroot(root1);access(root2);splay(root2);
}
int main()
{
   t[0].lx=t[0].rx=-2147483647;
   scanf("%d %d",&n,&m);
   for(int i=1;i<=n;++i)
   {
          scanf("%d",&t[i].col);
          t[i].lx=t[i].rx=t[i].col;
          t[i].num=1;
   }
   for(int i=1;i<n;++i)
   {
          scanf("%d %d",&a1,&b1);
          link(a1,b1);
   }
   for(int i=1;i<=m;++i)
   {
          scanf("%s",&q);
          if(q[0]=='Q')
          {
                    scanf("%d %d",&a1,&b1);
                    work(a1,b1);
                    printf("%d\n",t[b1].num);
          }
          if(q[0]=='C')
          {
                    scanf("%d %d %d",&a1,&b1,&c1);
                    change(a1,b1,c1);
          }
   }
   return 0;
}
```

有时，旋转标记会对解产生意想不到的影响，比如水果姐逛水果街。请用 **link-cut tree** 来做这道题，更加深刻的理解懒标记，以及更加熟练的维护复杂的区间信息。

水果姐逛水果街Ⅲ codevs3306

时间限制: 2 s

空间限制：256000 KB

题目等级 ：大师 Master

题目描述 Description

连续两天都没有被 cgh 难倒，水果姐心情很不错，第三天又来逛水果街。

今天 cgh 早早就来到了水果街等水果姐，因为他带来了帮手 cys。cgh 的魔法是把水果街变成树结构，而 cys 的魔法是瞬间改变某家店的水果价格。一边问一边改，绝不给水果姐思考的时间!

同样还是 n 家水果店，编号为 1~n，每家店能买水果也能卖水果，并且同一家店卖与买的价格一样。

cgh 和 cys 一共有 m 个操作。

操作的格式为

0 x y 获 1 x y

如果操作类型是 0，表示 cys 把第 x 家店的价格改为 y

如果操作类型是 1，则表示要求水果姐从第 x 家店出发到第 y 家店，途中只能选一家店买一个水果，然后选一家店（可以是同一家店，但不能往回走）卖出去。并且输出最多可以赚多少钱。

这题是给天牛做的。

输入描述 Input Description

第一行 n，表示有 n 家店

下来 n 个正整数，表示每家店一个苹果的价格。

下来 n-1 行，每行两个整数 x，y，表示第 x 家店和第 y 家店有一条边。

下来一个整数 m，表示下来有 m 个操作。

下来有 m 行，每行三个整数，表示一次操作。

输出描述 Output Description

每行对应一个 1 类型的操作，输出一个整数，表示面对 cgh 的每次询问，水果姐最多可以赚到多少钱。

样例输入 Sample Input

```
4
16 5 1 15
1 2
1 3
2 4
3
1 3 4
0 3 17
1 4 3
```

样例输出 Sample Output

```
15
12
```

数据范围及提示 Data Size & Hint

0<=苹果的价格<=10^8

0<n<=2*10^5
0<m<=10^5

代码：

```
#include<iostream>
#include<cstdlib>
#include<cstdio>
#include<cstring>
using namespace std;
const int MAXN=200005;
int stk[MAXN],n,m,top,a1,b1,q;
struct tree
{
   int ch[2],fa,maxx,mins,num,ans1,ans2,lazy;
}t[MAXN];
int max1(int x,int y)
{
```

```
    return x>y?x:y;
}
int min1(int x,int y)
{
    return x<y?x:y;
}
void pushdown(int root)
{
    if(t[root].lazy)
    {
        t[root].lazy=0;
        swap(t[root].ch[0],t[root].ch[1]);
        swap(t[root].ans1,t[root].ans2);
        t[t[root].ch[0]].lazy^=1;
        t[t[root].ch[1]].lazy^=1;
    }
}
void update(int root)
{
    if(t[root].ch[0])pushdown(t[root].ch[0]);
    if(t[root].ch[1])pushdown(t[root].ch[1]);
    t[root].maxx=max1(t[root].num,max1(t[t[root].ch[0]].maxx,t[
t[root].ch[1]].maxx));
    t[root].mins=min1(t[root].num,min1(t[t[root].ch[0]].mins,t[
t[root].ch[1]].mins));
    t[root].ans1=max1(max1(t[t[root].ch[1]].maxx,t[root].num)-m
in1(t[t[root].ch[0]].mins,t[root].num),max1(t[t[root].ch[0]].a
ns1,t[t[root].ch[1]].ans1));
    t[root].ans2=max1(max1(t[t[root].ch[0]].maxx,t[root].num)-m
in1(t[t[root].ch[1]].mins,t[root].num),max1(t[t[root].ch[0]].a
ns2,t[t[root].ch[1]].ans2));
}
bool isroot(int root)
{
    return t[t[root].fa].ch[0]!=root&&t[t[root].fa].ch[1]!=root;
}
void rot(int root)
{
    int fa=t[root].fa;
    int gfa=t[fa].fa;
    int t1=(root!=t[fa].ch[0]);
    int t2=(fa!=t[gfa].ch[0]);
    int ch=t[root].ch[1^t1];
    if(!isroot(fa))t[gfa].ch[0^t2]=root;
```

```
    t[ch].fa=fa;
    t[root].fa=gfa;
    t[root].ch[1^t1]=fa;
    t[fa].fa=root;
    t[fa].ch[0^t1]=ch;
    update(fa);
}
void splay(int root)
{
    stk[++top]=root;
    for(int i=root;!isroot(i);i=t[i].fa)
    {
        stk[++top]=t[i].fa;
    }
    while(top)
    {
        pushdown(stk[top--]);
    }
    while(!isroot(root))
    {
        int fa=t[root].fa;
        int gfa=t[fa].fa;
        if(!isroot(fa))
        {
            root==t[fa].ch[0]^fa==t[gfa].ch[0]?rot(root):rot(fa);
        }
        rot(root);
    }
    update(root);
    return;
}
void access(int root)
{
    int temp=0;
    while(root)
    {
        splay(root);
        t[root].ch[1]=temp;
        update(root);
        temp=root;
        root=t[root].fa;
    }
    return;
}
```

```
void changeroot(int root)
{
   access(root);splay(root);t[root].lazy^=1;
}
void link(int root1,int root2)
{
   changeroot(root1);t[root1].fa=root2;
}
void work(int root1,int root2)
{
   changeroot(root1);access(root2);splay(root2);
}
void change(int root,int d)
{
   splay(root);t[root].num=d;update(root);
}
int main()
{
   t[0].mins=1999999999;
   scanf("%d",&n);
   for(int i=1;i<=n;++i)
   {
      scanf("%d",&t[i].num);
      t[i].maxx=t[i].mins=t[i].num;
   }
   for(int i=1;i<n;++i)
   {
      scanf("%d %d",&a1,&b1);
      link(a1,b1);
   }
   scanf("%d",&m);
   for(int i=1;i<=m;++i)
   {
      scanf("%d %d %d",&q,&a1,&b1);
      if(q==0)
      {
         change(a1,b1);
      }
      if(q==1)
      {
         work(a1,b1);
         printf("%d\n",t[b1].ans1);
      }
   }
```

```
    return 0;
}
```

六、动态最小生成树

codevs1001 舒适的路线

2006 年

时间限制：2 s

空间限制：128000 KB

题目等级 ：钻石 Diamond

题解

题目描述 Description

Z 小镇是一个景色宜人的地方，吸引来自各地的观光客来此旅游观光。

Z 小镇附近共有

N(1<N≤500)个景点（编号为 1,2,3,…,N），这些景点被 M（0<M≤5000）条道路连接着，所有道路都是双向的，两个景点之间可能有多条道路。也许是为了保护该地的旅游资源，Z 小镇有个奇怪的规定，就是对于一条给定的公路 Ri，任何在该公路上行驶的车辆速度必须为 Vi。频繁的改变速度使得游客们很不舒服，因此大家从一个景点前往另一个景点的时候，都希望选择行使过程中最大速度和最小速度的比尽可能小的路线，也就是所谓最舒适的路线。

输入描述 Input Description

第一行包含两个正整数，N 和 M。

接下来的 M 行每行包含三个正整数：x，y 和 v（1≤x,y≤N，0 最后一行包含两

个正整数 s，t，表示想知道从景点 s 到景点 t 最大最小速度比最小的路径。s 和 t 不可能相同。

输出描述 Output Description

如果景点 s 到景点 t 没有路径，输出“IMPOSSIBLE”。否则输出一个数，表示最小的速度比。如果需要，输出一个既约分数。

样例输入 Sample Input

样例 1

```
4 2
1 2 1
3 4 2
1 4
```

样例 2

```
3 3
1 2 10
1 2 5
2 3 8
1 3
```

样例 3

```
3 2
1 2 2
2 3 4
1 3
```

样例输出 Sample Output

样例 1

```
IMPOSSIBLE
```

样例 2

```
5/4
```

样例 3

2

数据范围及提示 **Data Size & Hint**

N(1<N≤500)

M (0<M≤5000)

Vi 在 int 范围内

我把例题从魔法森林换成了这道题，原因是我觉得这道题反倒更适合讲…请自行完成 **NOI** 魔法森林。

简单的并查集

把边权按从小到大的顺序排好序枚举每一条边，把这条边的边权作为 **ans_max**，接下来从这条边开始按顺序（从大到小）枚举每一条比它小的边，并且把这条边的两端点划分到一个联通块中，并且判断起点和终点是否在同一个联通块中，如果在同一个联通块中，那么当前枚举到的边的权值就是 **ans_min**，然后更新当前最优解即可。

原理：枚举、贪心，因为我们要求 **ans_max** 与 **ans_min** 的最小比值，**ans_max** 与 **ans_min** 越接近肯定是越好的。所以我们枚举 **ans_max**,这样就可以把 **ans_max** 当成常数，那么 **ans_min** 的值一定是越大越好（当然，它不可能大过 **ans_max**），所以我们从当前枚举到的边开始，每次加一条略小的边进入，当 **S** 与 **T** 第一次联通时，**ans_min** 的值是最大的，那么就能保证他们的比值是当前 **ans_max** 下最小的。

时间复杂度：o(n^2logn)空间复杂度:o(n)

代码:

```
#include<iostream>
#include<cstdlib>
#include<cstring>
#include<cstdio>
#include<algorithm>
using namespace std;
const int MAXN=50005,MAXM=50005;
void unions(int x,int y)
int fa[MAXN],n,m,ans_max=2147483647,ans_min=1,s,t;
int findf(int x)
{
   if(x==fa[x])return x;
   return fa[x]=findf(fa[x]);
}
{
   if(findf(x)!=findf(y))
      fa[findf(x)]=y;
   return;
}
struct edges
{
   int p[2],vi;
}e[MAXM];
bool cmp(edges x,edges y)
{
   return x.vi<y.vi;
}
int gcd(int x,int y)
{
   while(y)
   {
      int t=y;
      y=x%y;
      x=t;
   }
   return x;
}
int main()
{
```

```
    scanf("%d %d",&n,&m);
    for(int i=1;i<=m;++i)
    {
        scanf("%d %d %d",&e[i].p[0],&e[i].p[1],&e[i].vi);
    }
    sort(e+1,e+1+m,cmp);
    scanf("%d %d",&s,&t);
    for(int i=1;i<=m;++i)
    {
        for(int j=1;j<=n;++j)
        {
            fa[j]=j;
        }
        for(int j=i;j;--j)
        {
            unions(e[j].p[0],e[j].p[1]);
            if(findf(s)==findf(t))
            {

    if(double(ans_max)/double(ans_min)>double(e[i].vi)/double(e
[j].vi))
                {
                    ans_max=e[i].vi;
                    ans_min=e[j].vi;
                }
                break;
            }
        }
    }
    if(ans_max==2147483647&&ans_min==1)
    {
        printf("IMPOSSIBLE\n");
        return 0;
    }
    int tmp=gcd(ans_max,ans_min);
    ans_max/=tmp;
    ans_min/=tmp;
    if(ans_min==1)
    {
        printf("%d\n",ans_max);
        return 0;
    }
    printf("%d/%d\n",ans_max,ans_min);
    return 0;
```

```
}
```

水水的 **spfa**

的用 **spfa** 可以求从起点到任意点的最大的最小边权...思路和上面差不太多，首先也是要枚举 **max** 作为边的限制条件，然后跑 **spfa...**

优化：如果我们从小到大枚举 **max** 就会发现，每次一跑都会以新加入的边为基础进行松弛的（废话）...如果不理解的话请自行跑两遍...那么就好说了，**dis** 数组每次不要清空，按边权枚举每条边，并且把边的两端点放入队列中如果跑完发现 **dis[t]**被更新了，那么说明至少经过了 **1** 条边权为 **max** 的边，所以最终的 **ans_max** 就是当前枚举到的 **max**，**ans_min** 就是当前的 **dis[t]**。

时间复杂度:**o(**开挂**)**数据太水...大概 **16ms**，理论上没这么好。空间复杂度:**o(n)**。

代码:

```
#include<iostream>
#include<cstdlib>
#include<cstring>
#include<cstdio>
#include<algorithm>
using namespace std;
const int MAXN=200005;
struct edges
{
   int to,next,vi;
}e[MAXN];
struct saves
{
   int from,to,vi;
}save[MAXN];
```

```
int
g[MAXN],n,m,mins,dmin[MAXN],tot,s,t,head,tail,q[MAXN],temp,ans
_max=2147483647,ans_min=1;
bool inq[MAXN];
void adds(int from,int to,int vi)
{
   e[++tot].to=to;
   e[tot].next=g[from];
   e[tot].vi=vi;
   g[from]=tot;
   return;
}
bool cmp(saves x,saves y)
{
   return x.vi<y.vi;
}
int max1(int x,int y)
{
   return x>y?x:y;
}
int min1(int x,int y)
{
   return x<y?x:y;
}
int spfa(int maxvi)
{
   while(head!=tail)
   {
      head=head%200001+1;
      int x=q[head];
      for(int i=g[x];i;i=e[i].next)
      {

   if(e[i].vi<=maxvi&&min1(dmin[x],e[i].vi)>dmin[e[i].to])
         {
            dmin[e[i].to]=min1(dmin[x],e[i].vi);
            if(!inq[e[i].to])
            {
               tail=tail%200001+1;
               q[tail]=e[i].to;
               inq[e[i].to]=true;
            }
         }
      }
```

```
            inq[x]=false;
        }
        if(temp!=dmin[t])
        {

        if(double(ans_max)/double(ans_min)>double(maxvi)/double(dmi
n[t]))
            {
                ans_max=maxvi;
                ans_min=dmin[t];
            }
            temp=dmin[t];
        }
}
int gcd(int x,int y)
{
    while(y)
    {
        int t=y;
        y=x%y;
        x=t;
    }
    return x;
}
int main()
{
    scanf("%d %d",&n,&m);
    for(int i=1;i<=m;++i)
    {
        scanf("%d %d %d",&save[i].from,&save[i].to,&save[i].vi);
        adds(save[i].from,save[i].to,save[i].vi);
        adds(save[i].to,save[i].from,save[i].vi);
    }
    scanf("%d %d",&s,&t);
    sort(save+1,save+1+m,cmp);
    for(int i=1;i<=n;++i)
    {
        dmin[i]=0;
    }
    dmin[s]=2147483647;
    for(int i=1;i<=m;++i)
    {
        head=tail=0;
        q[++tail]=save[i].from;
```

```
            q[++tail]=save[i].to;
            inq[save[i].from]=true;
            inq[save[i].to]=true;
            spfa(save[i].vi);
        }
        if(dmin[t]==0)
        {
            printf("IMPOSSIBLE\n");
            return 0;
        }
        int tmp=gcd(ans_max,ans_min);
        ans_max/=tmp;
        ans_min/=tmp;
        if(ans_min==1)
        {
            printf("%d\n",ans_max);
            return 0;
        }
        printf("%d/%d\n",ans_max,ans_min);
        return 0;
}
```

优美的 **LCT...**

如果强行**LCT**的话就会发现....和**noi2014**的魔法森林怎么**TM**那么像...简直就是一个题。

接着上面的思路...我们枚举**max**作为限制条件，每次寻找从**S**到**T**的一条路径，使得最小的边权最大...想到什么没有，当然是最大生成树，最大生成树上任意两点间简单路径上的最小边权就是**S**到**T**的最大的（最小边权）。于是我们从小到大枚举每一条边，如果把这条边的两点连上构成了一个环，则删除环上最小边并把这条边加入（如果这条边就是环上最小的边就不加），否则直接加入。然后判断**S**与**T**是否联通，如果联通就求**S**到**T**路径上的最小权值为

ans_min，枚举到的 **max** 作为 **ans_max**，更新最优解即可。

时间复杂度：均摊 **o(nlogn)** 空间复杂度：**o(n)**。

代码：

```
#include<iostream>
#include<cstdio>
#include<cstring>
#include<cstdlib>
#include<algorithm>
using namespace std;
const int MAXN=500005;
struct edges
{
   int p[2],vi;
}e[MAXN];
struct tree
{
   int fa,ch[2],num,mins,lazy;
}t[MAXN];
int
n,m,a1,b1,c1,stk[MAXN],top,ans_max=2147483647,ans_min=1,s1,t1;
int gcd(int x,int y)
{
   while(y)
   {
      int t=y;
      y=x%y;
      x=t;
   }
   return x;
}
bool cmp(edges x,edges y)
{
   return x.vi<y.vi;
}
bool isroot(int root)
{
   return root!=t[t[root].fa].ch[0]&&root!=t[t[root].fa].ch[1];
}
void update(int root)
{
   t[root].mins=root;
```

```
    if(t[t[t[root].ch[0]].mins].num<t[t[root].mins].num)
        t[root].mins=t[t[root].ch[0]].mins;
    if(t[t[t[root].ch[1]].mins].num<t[t[root].mins].num)
        t[root].mins=t[t[root].ch[1]].mins;
    return;
}
void pushdown(int root)
{
    if(t[root].lazy)
    {
        swap(t[root].ch[0],t[root].ch[1]);
        t[t[root].ch[0]].lazy^=1;
        t[t[root].ch[1]].lazy^=1;
        t[root].lazy=0;
    }
    return;
}
void rot(int root)
{
    int fa=t[root].fa;
    int gfa=t[fa].fa;
    int t1=(root!=t[fa].ch[0]);
    int t2=(fa!=t[gfa].ch[0]);
    int ch=t[root].ch[1^t1];
    if(!isroot(fa))t[gfa].ch[0^t2]=root;
    t[root].fa=gfa;
    t[root].ch[1^t1]=fa;
    t[fa].ch[0^t1]=ch;
    t[fa].fa=root;
    t[ch].fa=fa;
    update(fa);
    return;
}
void splay(int root)
{
    stk[++top]=root;
    for(int i=root;!isroot(i);i=t[i].fa)
    {
        stk[++top]=t[i].fa;
    }
    while(top)
    {
        pushdown(stk[top--]);
    }
```

```
    while(!isroot(root))
    {
        int fa=t[root].fa;
        int gfa=t[fa].fa;
        if(!isroot(fa))

    (root==t[fa].ch[0])^(fa==t[gfa].ch[0])?rot(root):rot(fa);
        rot(root);
    }
    update(root);
    return;
}
void access(int root)
{
    int temp=0;
    while(root)
    {
        splay(root);
        t[root].ch[1]=temp;
        update(root);
        temp=root;
        root=t[root].fa;
    }
    return;
}
void changeroot(int root)
{
    access(root);splay(root);t[root].lazy^=1;
}
void link(int root1,int root2)
{
    changeroot(root1);t[root1].fa=root2;
}
void cut(int root1,int root2)
{
    changeroot(root1);access(root2);splay(root2);t[root2].ch[0]
=t[root1].fa=0;
    update(root2);
}
int findroot(int root)
{
    access(root);splay(root);while(t[root].ch[0])root=t[root].c
h[0];
    return root;
```

```
}
void works(int root1,int root2)
{
   changeroot(root1);access(root2);splay(root2);
}
int main()
{
   scanf("%d %d",&n,&m);
   for(int i=1;i<=m;++i)
   {
      scanf("%d %d %d",&e[i].p[0],&e[i].p[1],&e[i].vi);
   }
   sort(e+1,e+1+m,cmp);
   scanf("%d %d",&s1,&t1);
   t[0].num=2147483647;
   for(int i=1;i<=m;++i)
   {
      if(findroot(e[i].p[0])!=findroot(e[i].p[1]))
      {
         t[i+n].num=e[i].vi;
         t[i+n].mins=i+n;
         t[e[i].p[0]].num=2147483647;
         t[e[i].p[1]].num=2147483647;
         link(e[i].p[0],i+n);
         link(i+n,e[i].p[1]);
      }
      else
      {
         works(e[i].p[0],e[i].p[1]);
         int ei=t[e[i].p[1]].mins;
         if(t[ei].num<e[i].vi)
         {
            cut(e[ei-n].p[0],ei);
            cut(ei,e[ei-n].p[1]);
            t[i+n].num=e[i].vi;
            t[i+n].mins=i+n;
            link(e[i].p[0],i+n);
            link(i+n,e[i].p[1]);
         }
      }
      if(findroot(s1)==findroot(t1))
      {
         works(s1,t1);
```

```
   if(double(ans_max)/double(ans_min)>double(e[i].vi)/double(t
[t[t1].mins].num))
          {
             ans_max=e[i].vi;
             ans_min=t[t[t1].mins].num;
          }
       }
   }
   if(ans_max==2147483647&&ans_min==1)
   {
      printf("IMPOSSIBLE\n");
      return 0;
   }
   int tmp=gcd(ans_max,ans_min);
   ans_max/=tmp;
   ans_min/=tmp;
   if(ans_min==1)
   {
      printf("%d\n",ans_max);
      return 0;
   }
   printf("%d/%d\n",ans_max,ans_min);
   return 0;
}
```

请用 **spfa** 和 **link-cut trees AC NOI** 魔法森林~