树链剖分及其应用

提要

- 树链剖分相关概念
- 树链剖分的实现
- 例题
- 总结

一、树链剖分的相关概念

剖分目的

- 树链就是树上的路径
- 剖分的目的是：树上路径信息维护。
  处理在树上进行改点问线，改线问点等一系列问题
- 将一棵树划分成若干条链，用数据结构(线段树，平衡树等)去维护每条链，复杂度为O(logN)。

剖分方法

- 盲目剖分
- 随机剖分
- 启发式剖分

综合比较，**启发式剖分**是剖分时的最佳选择。

轻边和重边

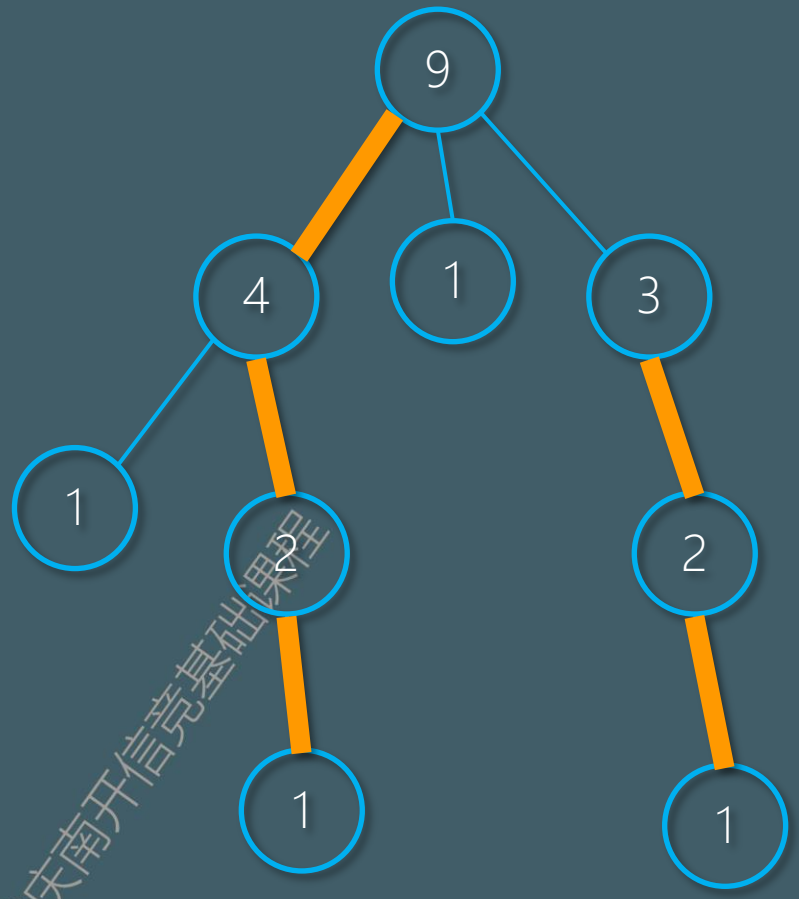
将树中的边分为：**轻边**和**重边**

定义size(X)为以X为根的子树的节点个数。

令V为U的儿子节点中size值最大的节点，那么边(U,V)被称为**重边**，树中重边之外的边被称为**轻边。**(称V为“**重儿子**”)

轻边和重边

下图节点圆圈中的数字表示该点的size值



- 粗边为重边。
- 另外，我们称某条路径为**重路径**(也叫**重链**)，当且仅当它全部由重边组成。

  比如：9,4,2,1是一条重路径

轻重边路径剖分的性质

- 轻边(U,V)，size(V)<=size(U)/2。
- 从根到某一点的路径上，不超过O(logN)条轻边，不超过O(logN)条重路径。

二、树链剖分的实现

重链剖分

- 重链剖分的过程为2次DFS
- 第一次：找重边
- 第二次：连重链(连重边成重链)

重链剖分

- 第一步，找重边:

  通过一次DFS，可记下所有的重边。

重链剖分：找重边


```
void Find_Heavy_Edge(int x,int father,int depth)        //当前讨论x号节点
{
    int i,child,MaxSize=0;
    Fa[x]=father;  Dep[x]=depth;  Size[x]=1;  Son[x]=0;
                                                //Son[x]记录x的重儿子的编号
    for(i=Last[x];i!=0;i=Next[i])               //这里采用边存储
    {
        child=End[i];
        if(child!=Fa[x])                        //因为采用边存储，某时刻child会指向x的父亲
        {
            Find_Heavy_Edge(child,x,depth+1);
            Size[x]+=Size[child];
            if(Size[child]>MaxSize)
            { MaxSize=Size[child];  Son[x]=child;  }
        }
    }
}
```

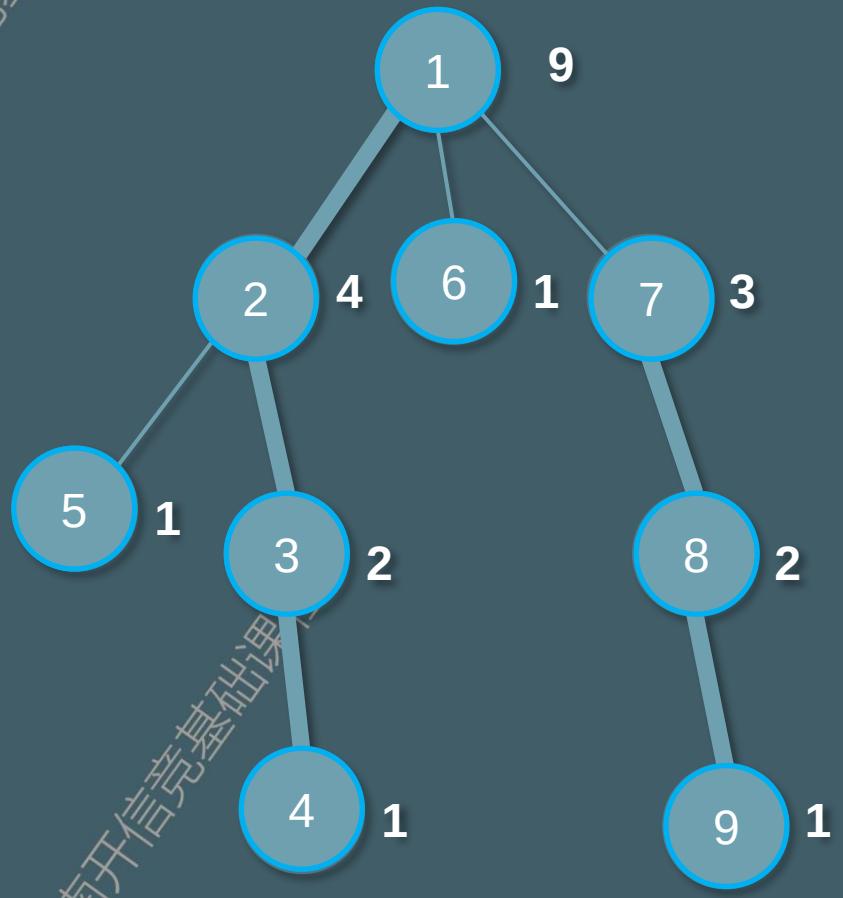
重链剖分

- 第二步，连重边成重链

  以根节点为起点，沿重边向下拓展，拉成重链。

  不在当前重链上的节点，都以该节点为起点向下重新拉一条重链。

重链剖分

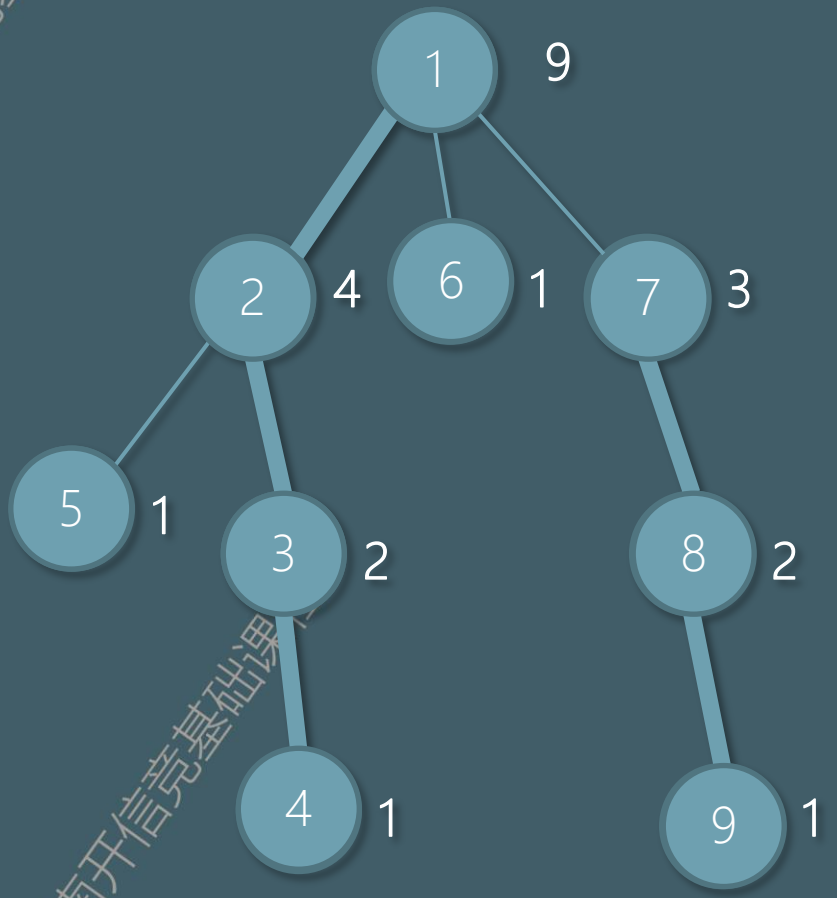


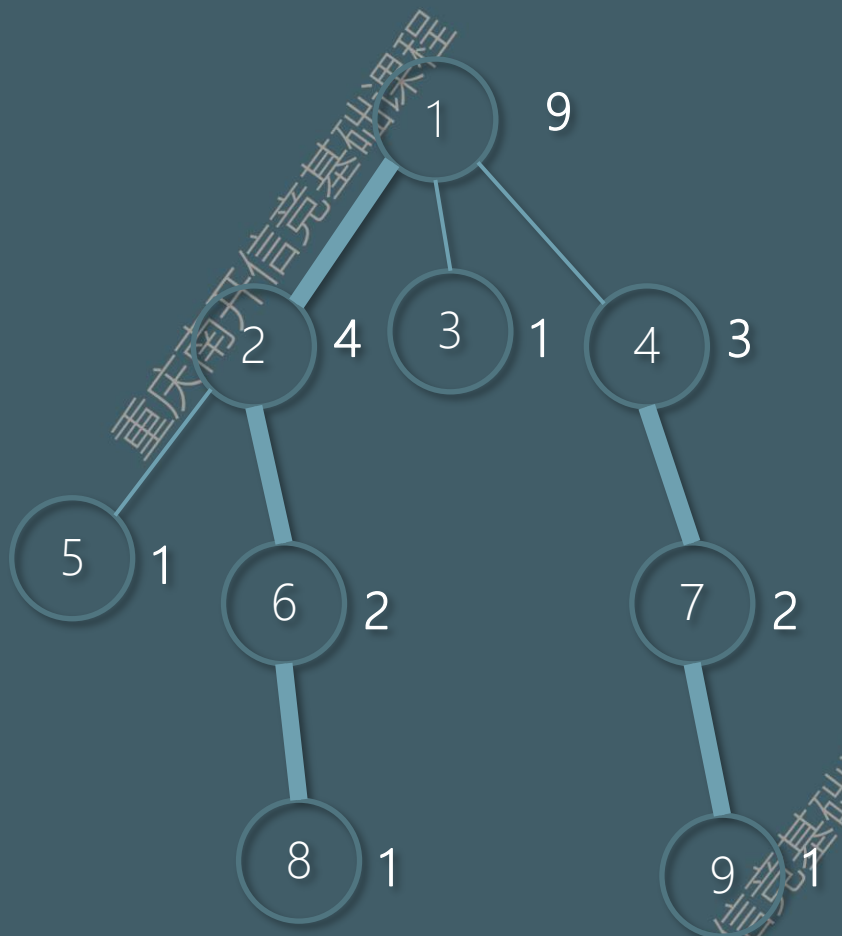
圈内的数字为原编号，圆圈旁边的数字为size[ ]值，粗线表示重边。

处理后，圈内数字代表每个点的新编号NewID。

重链剖分

链端top[ ]：记录每条重链的开始节点编号（原始编号）
重链上的点都有相同的top[ ]值





原始编号x	1	2	3	4	5	6	7	8	9
top[x]	1	1	3	4	5	1	4	1	4

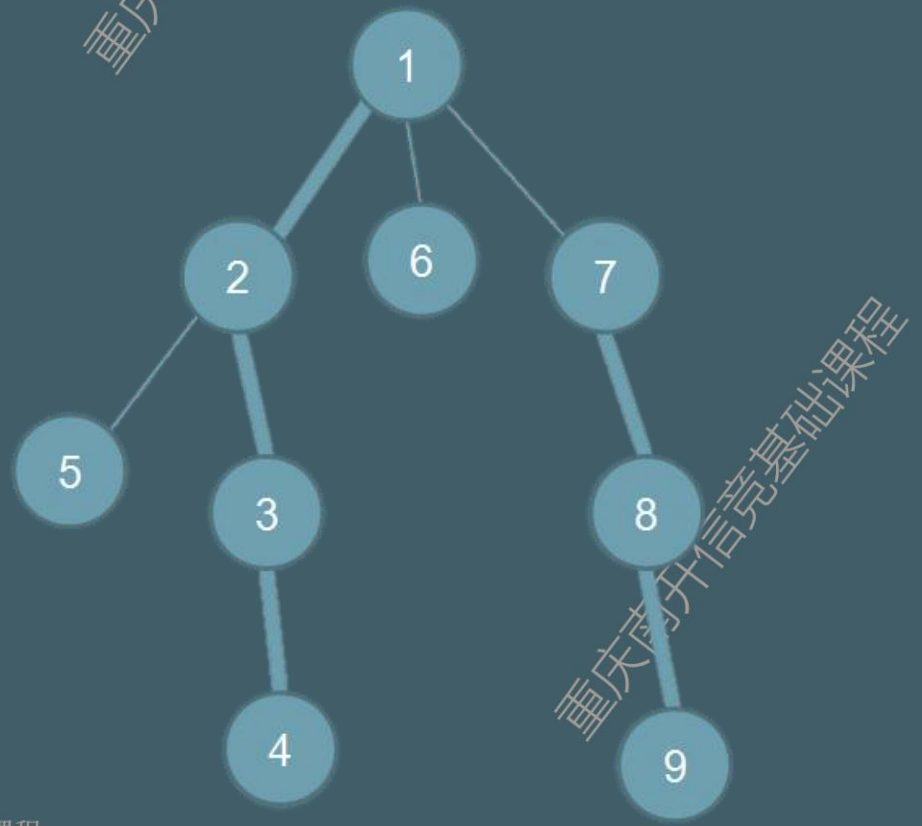
重链剖分：连重链

```
void Connect_Heavy_Edge(int x,int ancestor)      //ancestor意为祖先
{
    int i,child;                                 //Top[x]记录x所在重链的开始节点
    NewID[x]=++Label;    Top[x]=ancestor;        //全局变量Label用于给节点重新编号
    if(Son[x]!=0)Connect_Heavy_Edge(Son[x],ancestor); //x存在重儿子
    for(i=Last[x];i!=0;i=Next[i])                //将不是重儿子的点，从新拉出一条重链
    {
        child=End[i];
        if(child!=Fa[x]&&child!=Son[x])Connect_Heavy_Edge(child,child);
    }          //若child不在x所在的重链上，则从它从新拉出一条重链，重链的起点就是child本身
}
```

维护重链

- 剖分完之后，每条重链就相当于一段编号连续的区间，用数据结构(比如：线段树、平衡树等)去维护。
- 把所有的重链首尾相接，放到同一个数据结构上，然后维护这一个整体即可。

圈内为新编号
圈外为原始编号



修改操作

- 单独修改一个点的权值

  根据新的编号直接在数据结构中修改就行了。

修改操作

- 整体修改点 U和点V的路径上的权值

情况1： 如果U和V在同一条重链上

直接用数据结构修改NewID[U]至NewID[V]这段区间的值。

```
if(dep[v]>dep[u])swap(v,u);
Modify_SegTree(NewID[v],NewID[u]);
```

修改操作

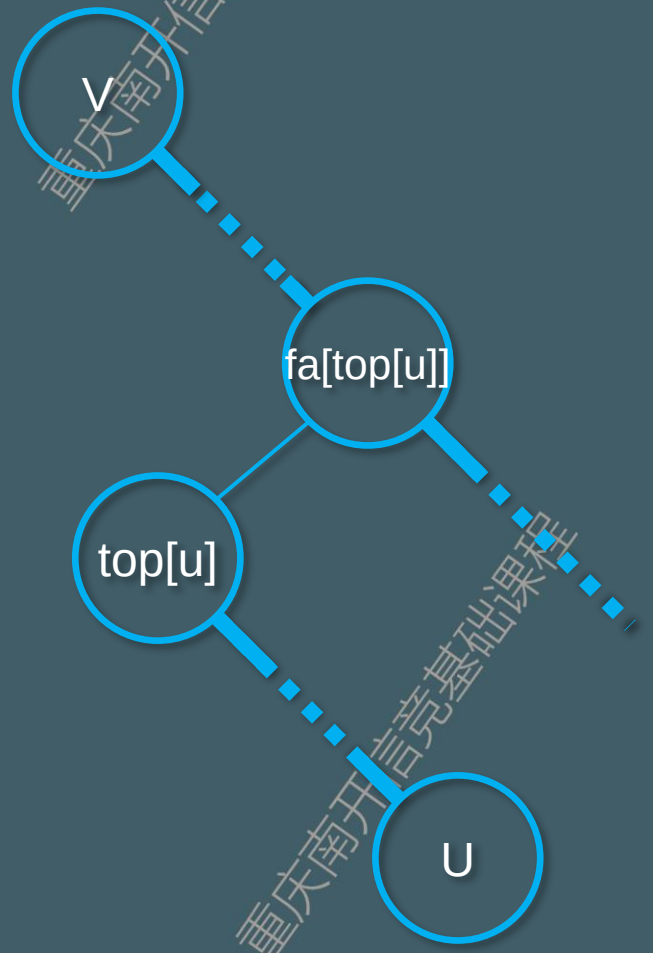
- 整体修改点 U和点V的路径上的权值

情况2：如果U和V不在同一条重链上

一边进行修改，一边将U和V往同一条重链上靠，然后就变成了“情况1”。

怎样将U和V向同一条重链上靠?

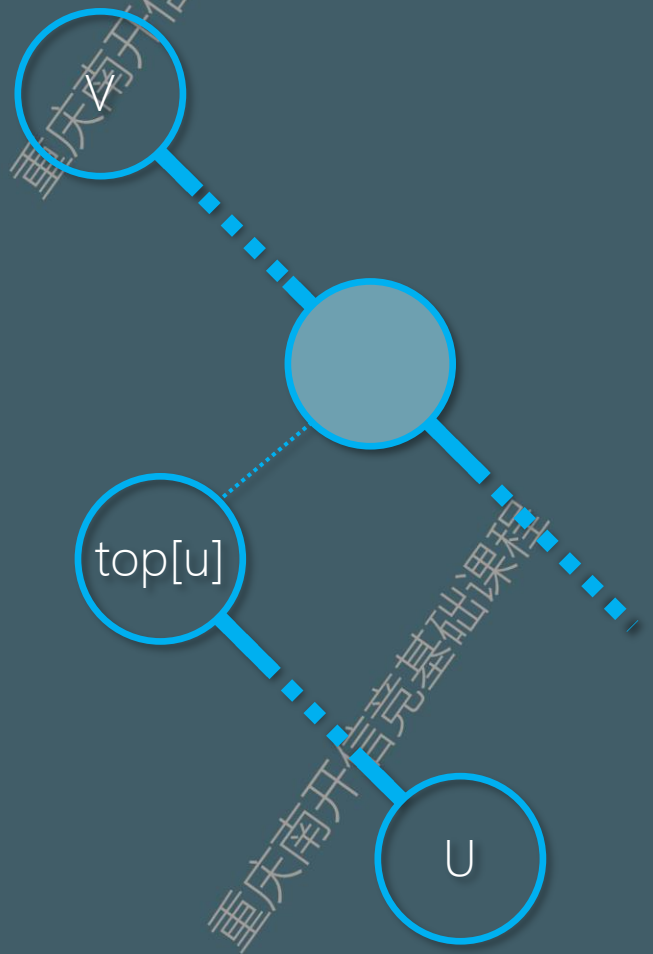
修改操作



- 情况A.若fa[top[U]]与V在同一条重链上。
- 修改点U与top[u]间的各权值，然后U跳至fa[top[u]]，就变成了情况1。

```
Modify_SegTree(NewID[top[u]],NewID[u]);
u=fa[top[u]];
if(dep[v]>dep[u])swap(v,u);
Modify_SegTree(NewID[v],NewId[u]);
```

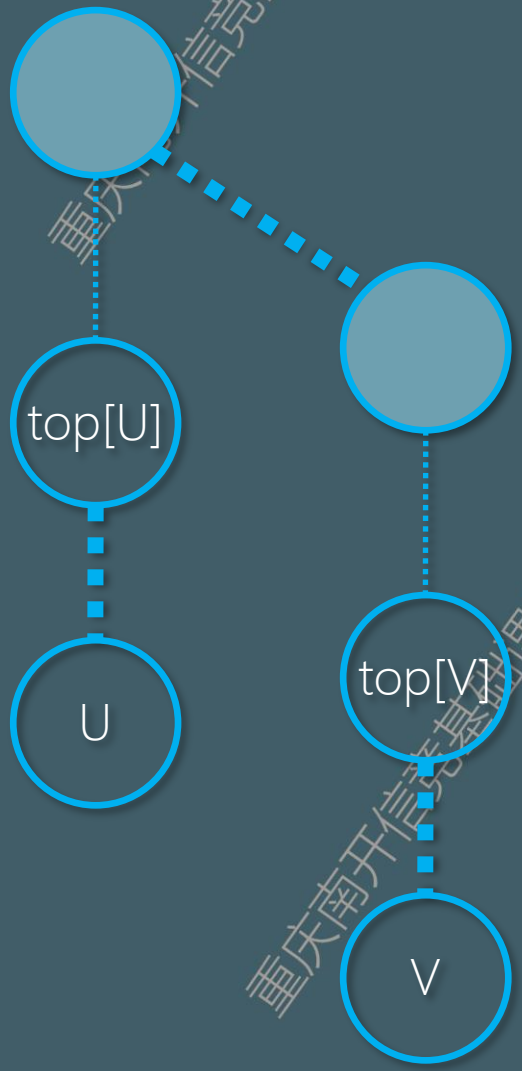
修改操作



- 情况B.若U向上经过若干条重链和轻边后才与V在同一条重链上。
- 不断地修改当前U和top[u]间的各权值，再将U跳至fa[top[U]]，直到U与V在同一条重链。

```
while(top[v]!=top[u])
{
    Change_SegTree(NewID[top[u]],NewID[u]);
    u=fa[top[u]];
}
if(dep[v]>dep[u])swap(v,u);
Change_SegTree(NewID[v],NewId[u]);
```

修改操作



- 情况C.若U和V都是向上经过若干条重链和轻边，到达同一条重链。
- 每次在点U和点V中，选择dep[top[x]]较大的点x，修改x与top[x]间的各权值，再跳至fa[top[x]]，直到点U和点V在同一条重链。

```
while(top[v]!=top[u])
{
      if (Dep[top[v]]<Dep[top[u]])swap(v,u);
      Modify_SegTree(NewID[top[v]],NewID[v]);
      v=fa[top[v]];
}
if(dep[v]>dep[u])swap(v,u);
Change_SegTree(NewID[v],NewId[u]);
```

修改操作

- 情况A、B是情况C的比较特殊的2种。
- “**情况1**”也只是“**情况2**”的特殊情况。
- 所以，这一操作只要用一个过程。

修改操作(代码模板)


```
void Change (int x, int y ,int d)      //将x到y路径上的所有点的权值修改为d
{
     while(top[x]!=top[y])
     {
          if (Dep[top[x]]<Dep[top[y]])swap(x,y);          //选深度大的点往上跳
           Modify_SegTree(NewID[top[x]],NewID[x],d);
                                  //修改存储重链的数据结构,将指定区间的值改为d
           x=fa[top[x]];                                  //x往上跳
      }
     if(Dep[x]>Dep[y])swap (x,y);
     Modify_SegTree(NewID[x],NewID[y],d);
}
```

Modify_SegTree(L,R,d)的作用如下：
为数据结构的修改操作：将区间[L,R]上的所有权值改为d,其中L比R更靠近根，即Dep[L]≤Dep[R]

查询操作

- 查询操作的分析过程同修改操作。
- 题目不同，选用不同的数据结构来维护值。

查询操作(代码模板)

```
int Query(int x,int y)          //比如查询x到y路径上点权总和
{
        int Sum=0;
        while(Top[x]!=Top[y])
        {
                if(Dep[Top[x]]<Dep[Top[y]])swap(x,y);
                Sum+=Query_SegTree(NewID[Top[x]],NewID[x]);
                x=Fa[Top[x]];
        }
        if(Dep[x]>Dep[y])swap(x,y);
        Sum+=Query_SegTree(NewID[x],NewID[y]);
        return Sum;
}
```

求LCA

- 若两个点处在同一条重链上，深度低的那个就是LCA。
- 如果不在同一条重链上，那么就跟前面谈的修改操作一样，让两个点向上跳，直至它们在一条重链上

求LCA(代码模板)

```
int LCA(int x,int y)
{
        while(Top[x]!=Top[y])
        {
                if(Dep[Top[x]]<Dep[Top[y]])swap(x,y);
                x=Fa[Top[x]];
        }
        if(Dep[x]>Dep[y])swap(x,y);
        return x;
}
```

小小总结一下

其实树链剖分就是把树的边映射到线段树上的数据结构。

实现的话很简单，用两个dfs处理数数的信息，重边以及轻边，然后就是一些线段树的操作了。