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

- 树路径信息维护。
- 将一棵树划分成若干条链，用数据结构去维护每条链，复杂度为O(logN)。

剖分方法

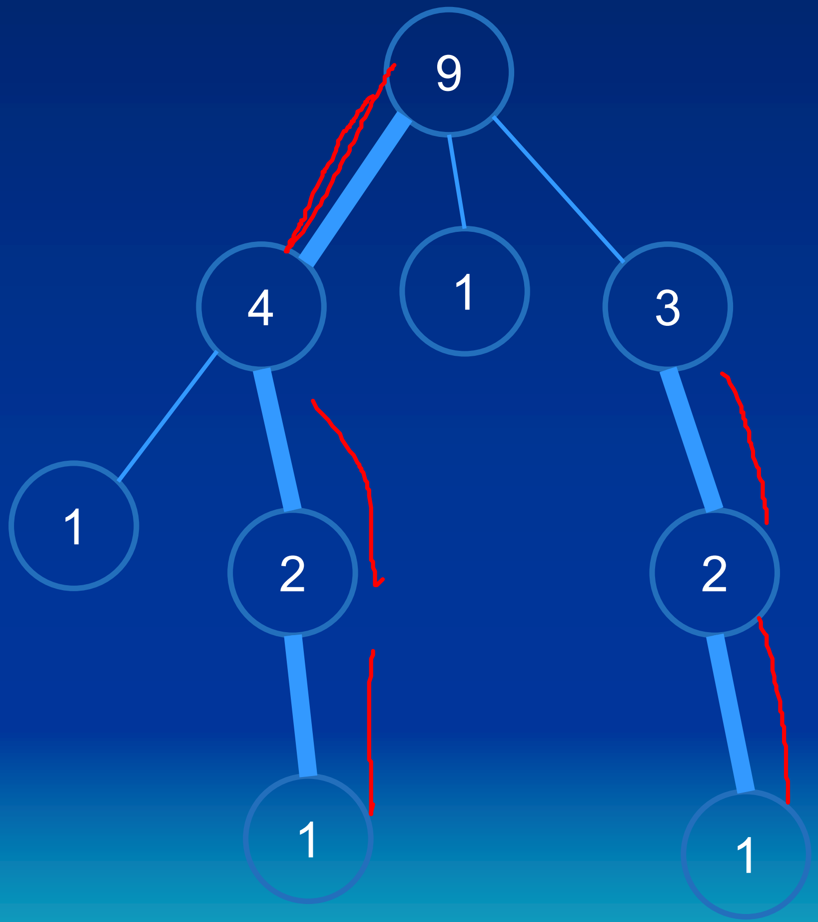
- 盲目剖分
- 随机剖分
- 启发式剖分

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**。

轻边和重边



- 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。

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

树链剖分

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

树链剖分

- 找重边

  一次DFS，可记下所有的重边。

  – 用一次DFS或BFS求出每个节点的尺寸(即自己的子孙节点个数+1)，然后每一个节点选择一个尺寸最大的子节点(有多个最大择随便选择一个)，连接该节点与这个子节点的边化为重边。

树链剖分

```
FIND-HEAVY_EDGE (x, father,depth)
    father→fa[x], depth→dep[x]
    1→size[x],0→maxsize,0→son[x],…
     while there exists a child of x not be visted
         do FIND-HEAVY_EDGE(child,x,depth+1)
             size[x]+=size[child]
             if size[child]>maxsize
             then size[child]→maxsize
                   child→son[x]
```

树链剖分

- 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。

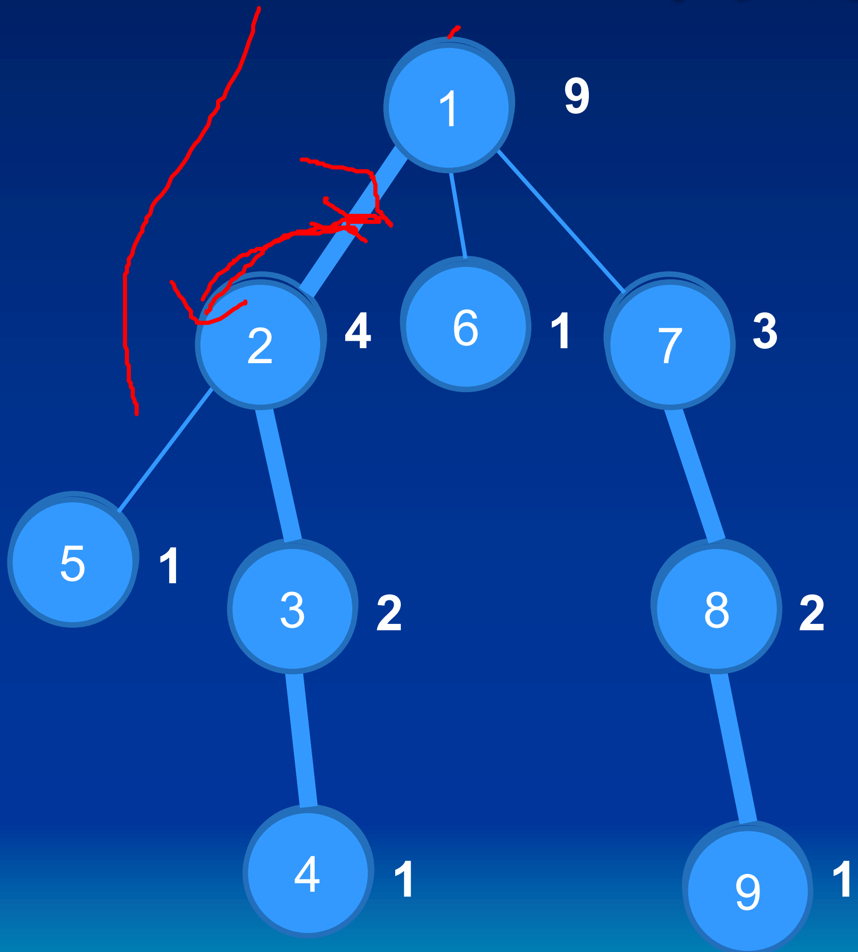
  于是每一个节点都属于且仅属于一条重链

树链剖分

```
CONNECT-HEAVY_EDGE (x, ancestor)
    ++label→tid[x], ancestor→top[x],…
    if son[x]≠0
    then CONNECT-HEAVY_EDGE(son[x],ancestor)
    while there exists a child of x not be visted
     do CONNECT-HEAVY_EDGE(child,child)
```

说明：top[x]表示：节点x所在重链的根

树链剖分



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。

圈内数字代表每个点的新编号tid。

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

维护重链

- 剖分完之后，每条重链就相当于一段区间，用数据结构去维护。
- 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。

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

  1、如果U和V在同一条重链上

  直接用数据结构修改tid[U]至tid[V]间的值。

修改操作

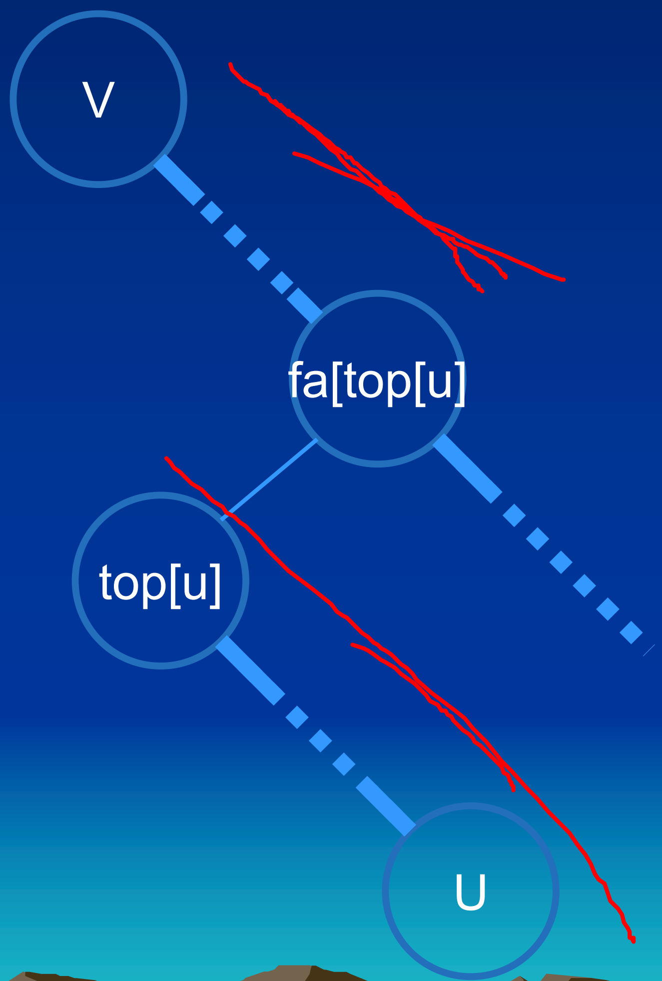
- 整体修改点 U和点V的路径上的权值

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I的情况。

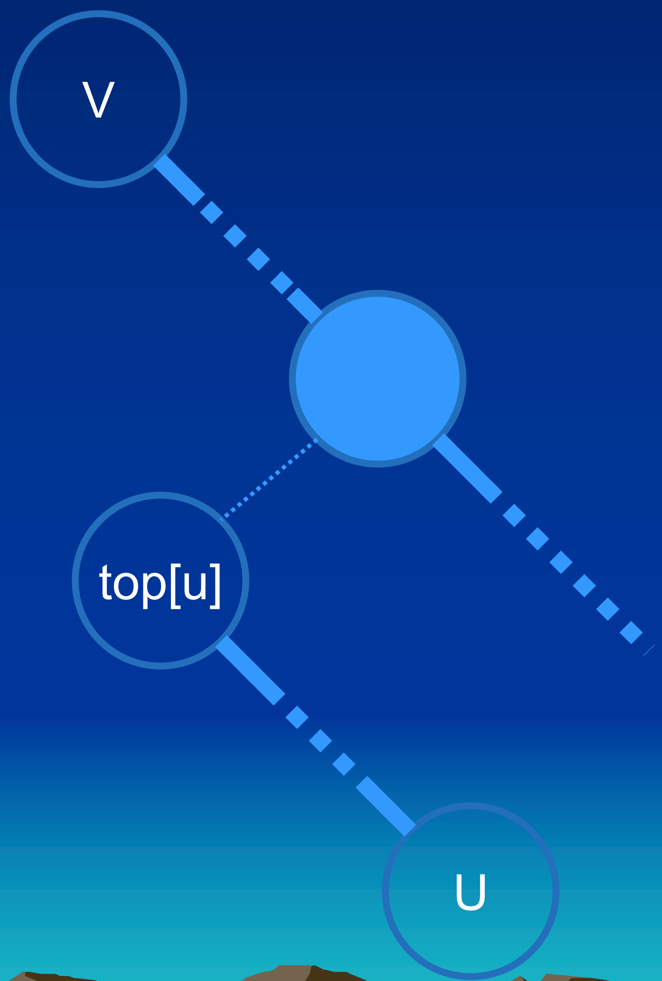
怎样将U和V向同一条重链上靠?

修改操作



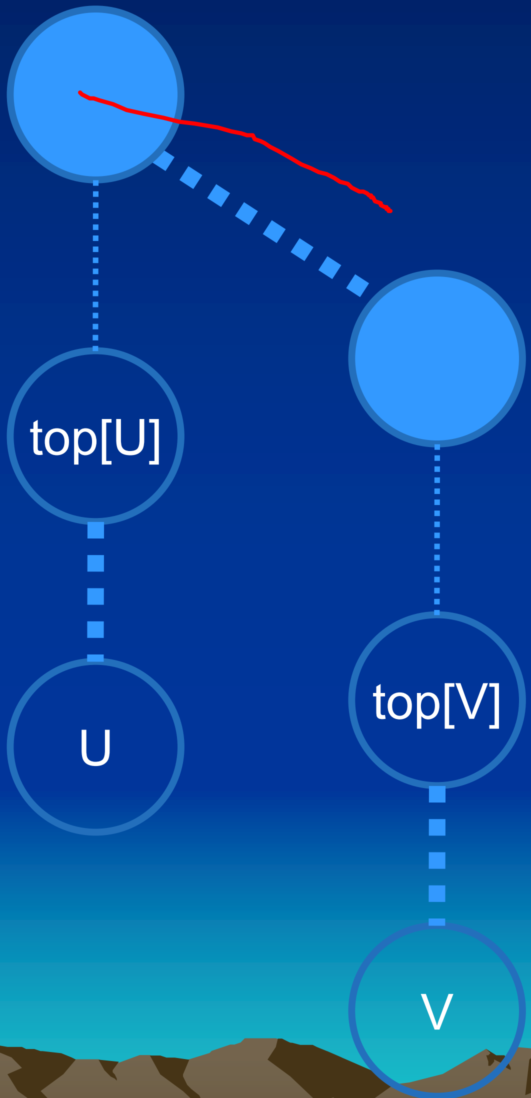
- A.若fa[top[U]]与V在同一条重链上。
- 修改点U与top[u]间的各权值，然后U跳至fa[top[u]，就变成了I的情况。

修改操作



- B.若U向上经过若干条重链和轻边与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。

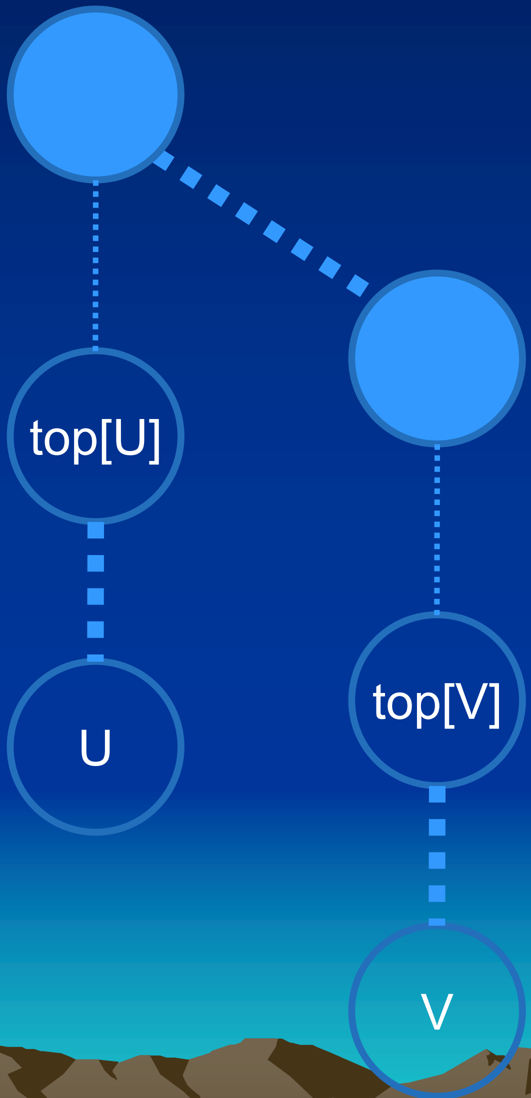
修改操作



- 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。有两种方法：
- ①可先求得U和V的最近公共祖先LCA，操作就变为整体修改U到LCA和V到LCA值。

- 求LCA的算法每次询问只要不超过O(log^2 n)的复杂度,就不会影响整个算法的复杂度.这里,推荐大家使用**倍增算法[O(n log n)-O(log n)]**,比Tarjan算法好写。效率也较高。

修改操作



- 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。有两种方法：
- ②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。

修改操作

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

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。

三、例题

Query on a tree

- 题意：

  一棵N(N<=10000)个节点的树，每条边都有一个权值，要求进行两种操作：

  CHANGE i ti：改变第i条边的权值为ti

  QUERY a b：询问节点a和节点b之间的路径中权值最大的边的权值

- 时限：5s

Query on a tree

- 样例

输入:

数据组数

1

3

N

1 2 1

2 3 2

QUERY 1 2

CHANGE 1 3

QUERY 1 2

DONE

a b c:表示节点a和节点b之间有一条权值为c的边

输出:

1

3

对于每一个QUERY操作输出对应结果

Query on a tree

单纯的模拟?

- 直接模拟操作，修改操作每次可以在O(1)做到，询问操作期望复度为$O(\log_2 N)$，不过最坏的情况下会达到O(n)。
- 显然，在大量询问的情况下，直接模拟操作会TLE。

Query on a tree

树链剖分?

- 这是一道树链剖分的经典入门题。
- 建立树，进行重链剖分，用数据结构维护重链，再进行修改操作和查询操作。
- 比较理想的支持修改操作且维护区间最大值的数据结构是线段树。

Query on a tree

- 不过，题中需要维护的是边权。
- 将边(fa[x],x)的权值，作为点x的权值。
- 根节点权值为一个无限小的值。

四、总结

总结

- 树链剖分在各类比赛中越来越多的被考到，也是解题很有用的工具。
- 平时多写写，容易有感觉，提高正确率。
- 注意学会缩代码和提高程序的效率。
- http://blog.csdn.net/acdreamers/article/details/10591443
- http://blog.sina.com.cn/s/blog_7a1746820100wp67.html
- http://www.bilibili.com/video/av4482146/