浅谈动态树的相关问题及简单拓展

长沙市雅礼中学 黄志翱

摘 要

动态树是在OI中很常见的一种问题。解决各种动态树问题的算法：树链剖分，树分治和link/cut tree已被广泛了解和学习。同时，用树的dfs序或者ETT[1]维护子树信息的方法也已经众所周知。本文将对相关问题进行总结，并提出一些原创性的解法。在最后，本文将介绍LCT的两个拓展：top tree和link/cut cactus，用来解决相关问题。

1 动态树问题

动态树问题(Dynamic Trees)是指在树上动态维护相关信息的问题。不过有时候在提到动态树时，会专指LCT。

在一般的动态树问题中，会出现如下操作：

1. 添加一条边或者删除一条边，维护树的形态。
2. 将树上的一条路径上的权值进行一定操作，维护每个点的权值。
3. 查询树上一条路径上的信息。
4. 给树的一棵子树的权值进行一定操作，维护树上每个节点的权值。
5. 查询子树信息。

[1]欧拉遍历树

2 古老的动态树问题

算法竞赛中出现过各种各样的和树有关的题目。解决这些问题的方法通常也是使用各种树型结构——比如说平衡树。

在一些比较古老的题目中，树的形态是相对固定的，这使得解题者不用将注意力花费在维护树的形态，更多地是如何将树上的信息按照某种形式组织，从而实现维护。

2.1 树链剖分

树链剖分的核心是将树上的边按照一定规则划分为轻边和重边。将重边形成的链用相关数据结构进行维护。可以证明一个点到根的路径至多被分成$O(\log n)$条重边和$O(\log n)$条轻边。于是就可以在$O(\log n *$单次询问复杂度)完成所有操作。

2.2 点分治

点分治是通过寻找树的重心将树进行分开，从而做到了将树的大小在每次分治后能够减少一半。

通常点分治解决的，是和树上的路径，或者点对有关的问题。在朴素的做法中，经常能够通过枚举路径两个端点之间的LCA，从而很方便地统计相关路径信息。而在点分治的做法中，只需要枚举在分治时选择的重心，可达到同样的效果。

3 Link/cut tree

3.1 什么是LCT

可以参考4中对LCT的介绍。

具体来讲，动态树就像是用Splay维护所有实边的树链剖分。利用动态树的access操作，可以取出一个点到根的链。通过对Splay 的翻转操作，可以实现换根。利用两次access可以取出两个点的LCA。

在均摊分析下，单次操作是均摊$O(\log n)$。于是可以解决相当多的问题。

3.2 边的信息与点的信息

在有些题目需要对边信息和点信息进行维护，解决这个问题的方法通常有两种：

第一种是将一条边(u,v)，新建一个点u'，将这条边变成连接$(u,u'),(u',v)$的两条边，将相应的边的信息存到点上。

第二种方法是同时记录一条链与每个点关联的两条边。在换根的时候进行相应的维护。

3.3 如何使用LCT

考虑一个最简单的问题：

1. 每次可以添加一条边，或者删除一条边。
2. 给一条链上每个点加上一个数。
3. 询问一条链的权值和。

考虑怎么用动态树实现：

1. 加边：将一个点作为它所在的树的根，并设置一条它连往其父亲的虚边。
2. 删边：对边的一个端点进行access操作，将这条边变成虚边，直接删除。
3. 给链加上一个数：将链上一个端点设为根，access另一个点，使得整条链在一个Splay中。利用Splay将整棵树加上一个权值。
4. 查询链上的权值：同上得到链所对应的Splay，在平衡树中查询整棵树的权值和即可。

可以发现，只要是能够用平衡树维护的序列问题，很容易通过LCT推广到树上。比如说将相应的区间加减设为区间赋值。查询的也不一定是链上的和，还可以是链上的最大值或者许多其它信息。

3.4 LCT的简单应用

3.4.1 明显的动态树问题

很多问题直接使用动态树即可。比如说BZOJ 3091 城市旅行。

此题要求维护树，支持删边加边，链上的权值加减，询问最大子段和。

显然可以使用LCT维护出树的形态，接下来只需要考虑区间赋值和查询子段和的操作了。

最大子段和可以使用维护最大前缀和以及最大后缀和的方法，利用相关信息合并解决。做法很经典，就不细说了。

3.4.2 隐藏的树结构

有很多问题并没有显式地将树告诉解题者，但经过分析之后可以发现明显的树结构，故而可以用动态树进行解决。比如IOI 2011 elephant 一题。只要注意到了每个点只有一个唯一的父亲等类似的树的特性，这些问题就迎刃而解了。

3.5 LCT在一些简单图论问题中的应用

树是描述图的重要工具，具体讨论可以参考3。动态树的灵活使得可以用其来解决一些图论问题。

3.5.1 只有加边的最小生成树

最小生成树是经典问题。而动态维护最小生成树则是一个相当困难的问题。

如果使用分治算法，可以做到$O(m \log m)$的复杂度。具体参考5的解题报告。但是这个做法是基于离线的，强制在线的话，只有一些复杂度不是很优美，相当复杂的算法。

但是如果只存在加边操作，且强制在线维护的话，就可以用动态树维护了。最小生成树的环切性质保证了只要每次在加入一条边之后，删除环上权值最大的边，则仍然是最小生成树。而查询路径最大值是动态树基本操作，问题解决。

3.5.2 最小极差路径

给定一个带权无向图，求从S到T的一条路径，使得路径上的最大权值减最小权值最小。

这个问题可以用动态树解决：将边排序，从小到大枚举边的最大值，实际上要求的是小于当前最大值的边中最小权值最大的一条路径。最小权值最大其实是最小瓶颈生成树的问题的一个子集，而最小生成树一定是最小瓶颈生成树，故而可以套用之前的做法解决。

3.5.3 动态维护图是否是二分图

给定一张图，可以加边或删边。每次求它是不是二分图。可以离线。

此题类似离线动态维护图的连通性，可以用分治做，具体不再赘述。

离线处理出每条边的删除时间，维护关于删除时间的最大生成树，也就是加一条边之后若形成环，则删掉环上的最早删除的边。删除边时还要做一件事：询问这条边与树边是否形成奇环，如果是则加入一个集合。删边时如果是树边直接删，否则若在集合中则从集合中删除。如果某个时刻集合为空则是二分图，否则不是二分图。正确性不是很显然，但可以证明。

而所有操作直接用动态树维护即可。

4 dfs序和ETT

可以发现，之前提到的动态树问题几乎都是和链上信息有关的。因为LCT本身就是用链维护树的结构。不过有很多问题是和树上的子树信息有关，单纯地用动态树解决会有一定困难。

这时候就有了另一种强大的工具，树的dfs序列——在dfs中，只有遍历完了一个点的子树才会继续访问其它点。故而，将点按照dfs 中的顺序写出，一个点的子树总是一个连续的区间。这样就可以使用各种数据结构进行维护了。

4.1 简单的子树维护问题

很多问题都是具有很明显的子树查询操作，可以用dfs序直接解决。

4.1.1 树

题目来源：IOI2012中国国家队训练罗雨屏

题目大意是给定一棵树，要求支持：

1. 换根
2. 修改点权
3. 查询子树最小值

经过简单分析后可以发现，这题从LCT入手并不明智。虽然有换根这个很灵活的操作，但树的形态并不会改变。

不妨使用树的dfs序。修改点权使用线段树维护区间最小值。

那么换根和查询怎么办呢？很容易发现，无论根在哪儿，查询的，要么是这个点在根为1时的子树，或者这棵子树以外的其它点。对于前者，直接查询区间最小值，否则，查询这个区间的补集的最小值，问题解决。

4.2 平衡树对dfs序的维护

得到一棵树的dfs序之前必须对整棵树进行dfs，如果树的形态会改变会怎么样呢？

可以发现，如果固定根，在加入一个点或者删除一个点后，在dfs序中相当于插入和删除一个数，只要能够找到这个数的位置就可以用平衡树维护，而这是非常容易的事情。

同理，如果是将一棵子树加入和删除，对应在dfs序中的改变只不过是相应的区间插入和删除，使用Splay或者没有旋转的Treap即可。

有了平衡树。就可以解决一些相对不是那么简单的题目。

4.2.1 Mashmokh's Designed Problem

题目来源：Codeforces Round 240 Div1 E

题目给定一棵有根树，且每个点连出去的边都有顺序。操作有：1.查询两个点之间的距离，2.以点u为根的子树从树中分开，并添加一条其与其某个祖先的连边，作为该祖先的最后一个儿子，3.查询从一个点出发按边的顺序进行dfs，深度为k的最后遍历的点。

这题使用LCT不是特别好解决，但是从dfs序的角度思考问题非常显然。删边和加边操作等价于区间操作。接下来只需要解决这么些问题：

1. 查找一个深度为k的在dfs序中的最后面的元素：注意到在dfs序中点的深度一定是连续变化的。维护一段区间中的深度最深的点和深度最浅的点，即可判断出该段区间是否存在深度为k的点，故而可以二分查找最后出现的深度为k的位置。

2. 利用上述方法亦可以查询一个点任意深度的祖先。

3. 在第一种询问中要查询u, v两个点的LCA，使用倍增可以做到$\log^2 n$的复杂度。考虑dfs序中在u和v之间的深度最小的点，其父亲就一定是LCA，转化为区间最小值，在$O(\log n)$解决。

所以本题就解决了。

4.2.2 dfs序与可持久化

由于使用了平衡树维护dfs序。而平衡树是可以可持久化的。故而有了这题：

题目来源：毛啸

给定一棵以0号点为根的树，要求支持以下操作：

1. 在树上添加一个点，权值为0

2. 将一棵子树的权值统一加上一个值

3. 询问一个点的权值

4. 询问一棵子树的权值和

5. 将当前树的状态变为某次操作后的树的状态

子树权值加减，加点和子树查询即区间查询都是非常基础的平衡树操作。

但是一旦要求可持久化后，就给问题制造了困难。如果要查询一个点对应的区间，就必须沿着这个点的父亲进行查询，而如果在平衡树中维护父亲，是

不可能做到可持久化的。除非将内存池进行持久化，但这仅限理论分析，在实际中无论是常数还是编程复杂度都无法接受。

必须使用不维护父亲的平衡树，但又必须查询到指定的点。为了解决这个问题，故在解法中引入了动态标号的概念：

给平衡树中的每个点维护一个标号，再对标号和平衡树同时进行可持久化。为了维护标号，需要使用到重量平衡树，参见6。具体来讲，如果选择使用Treap来维护持久化平衡树，在插入一个点之后，修改该子树中所有的标号。由于重量平衡树的性质，子树的期望大小为$O(\log n)$。为了将该标号持久化，需要再使用一个可持久化数组，如果用线段树实现，时间复杂度为$O(n \log^2 n)$。

4.3 欧拉遍历

下面将介绍一下欧拉遍历树，即ETT。

4.3.1 什么是欧拉遍历

设T为一个无向树。将T中的每条边拆成两条有向边，然后任选一个节点S作为起点进行DFS，以上过程称为对树T的欧拉遍历。注意，以任一节点作为起点进行欧拉遍历都可能的到相同的结果，因为欧拉遍历对应的是一个环。

4.3.2 欧拉遍历序列

在欧拉遍历中经过的有向边形成的序列。

4.3.3 欧拉遍历树

维护欧拉遍历序列的二叉搜索树。一般用Splay 或Treap实现。

欧拉遍历树注重的是对一棵树整体的信息进行维护。与DFS序不同的是，它可以维护以任意节点作为根时所有子树的信息。故而，可以用来实现换根这个对于dfs序有点困难的问题。

4.3.4 一些实现的细节

实际上只要支持两个操作即可：加边和删边。

1. 加边（连接两颗树）：可以看做是将两个环拼在一起。先把两个环拆开再拼到一起即可。注意两环之间拼接的位置。

2. 删边（将一棵树分成两颗树）：可以看做是将一个环分成两个环。这个操作也不难。

4.4 ETT的简单应用

给定一张图，每次可以加入一条边，删除一条边，查询任意两个点的连通性[2]。

这个问题的做法比较复杂，这儿只做一些简单描述。

因为要求的是两点之间是否联通，不由而然会联想到生成森林。但是这个做法无法解决删边的问题。

故而使用分治算法：给每条边一个标号e_i，使得标号小于等于e的边形成的连通块的大小小于等于2^e。可见$\max(e)$为$\log n$。用ETT对于所有e，维护标号小于等于e的边关于标号的最小生成树。

加边的话，将边的标号设为$\log n$，加入该层的最小生成树，时间复杂度为$\log n$。

询问连通性，在第$\log n$层查询两点的连通性，时间复杂度$\log n$。

最关键是删边。首先找到这条边的标号，如果它不在当前标号的最小生成树中，则直接删除，复杂度依旧为$O(\log n)$。

否则，设边的两个端点为u, v，在所有大于等于这条边的标号的图中寻找连接u, v所对应的子树的边：找到u, v所对应的子树，设u的子树大小小于v的子树大小。暴力扫描u这棵子树向外连接的所有当前标号的边，一旦和v所对应的子树相连，就找到了替代边，否则，将这些边的标号减1，加入当前标号-1的图中。因为u子树的大小必然是原来子树大小的一半以下，所以不会破坏分治的性质。

这样，标号减1的操作次数就是总时间复杂度，而标号的总和为$O(m \log n)$，总时间复杂度为$O(m \log^2 n)$。

因为要维护子树大小，进行删边和加边操作，以及查询一棵树中连接的当前标号的边，故而可以使用ETT实现。

[2]经典问题

5 LCT的一些扩展

LCT不仅仅可以完成简单的链操作，还可以实现一些比较有趣的问题。这儿举两个例子进行说明。

5.1 LCT与子树信息维护

在前面的内容中，用dfs序和ETT解决了子树询问问题，但是是否LCT就完全不能解决子树问题呢？

比如说4.1.1这题，难道就没法使用LCT实现么？其实是可以的。

不妨考虑在动态树对于每个点维护这样的信息：这个点连出去的所有虚边所对应的子树中权值的最大值（为了方便，通常会包括自己）。如果知道了每条虚边对应的子树中的权值的最大值，就可以使用堆进行维护。记这个值为A值。

同样的，可以定义一条链上的最大值为链上所有点连出去的虚边的最大值，即所有点的A值的最大值，这个在Splay中可以很容易维护。这样，查询以链上某个点为根的子树中的权值最大值，只需要在Splay中查询区间中最大的A值。

接下来考虑A值是如何变化的，动态树最关键的操作是access。在access的一次操作中，会将一条实边切换成虚边，将一条虚边切换为实边。在实边变成虚边时将儿子对应的子树最大值加入父亲的A值中，反之则删除。随着access的进行，相应地会修改点到根的路径上的所有信息。不过由于LCT保证了均摊$O(\log n)$的复杂度，再乘上维护A值所使用的堆的复杂度，问题在均摊$O(n\log^2 n)$的复杂度内解决。

关于换根操作，在此做些额外的说明，A值的维护是和根完全无关的，故而无需担心因为换根操作而使得这个做法失效。

5.2 LCT与链翻转

这个问题来自于这么一道题：

5.2.1 Lord

题目来源：IOI2012中国国家队训练陈文潇

求一个子树中的数的和。要求支持以下操作:

1. 查询以x为根，y子树中的所有数的和。

2. 将x,y的路径上的数翻转。

3. 将x 的值改为y。

4. 将x,y相连。

5. 将以x为根,把y向其父亲的边删除。

这题只有询问子树和的操作。但是由于链翻转的操作，从而不能够使用ETT。只能考虑动态树。

使用之前提到的做法，确实可以维护子树信息，但是怎么实现将路径上的权值进行翻转呢？

不妨考虑这么一种做法：并不在动态树上维护权值，而是对于动态树上的每一条链用一棵新的Splay维护其权值，这样翻转操作就可以实现了。再在动态树上对每个点维护其连出去的虚边的子树权值和即可解决问题。

初看之下复杂度是$O(n\log^2 n)$，但实际上两种Splay操作是等价的，不会破坏均摊分析，总时间复杂度应该依旧是动态树的$O(n\log n)$。

6 sone1

6.1 题目大意

sxyz里有一群神犇。要求对树维护：

1. 将某棵子树中所有点的权值加上或变成指定的数。

2. 将某条链上所有点的权值加上或变成指定的数。

3. 查询子树中的权值最大值，最小值，或者权值和。

4. 查询某条链上所有点的权值的最大值，最小值，或者权值和。

5. 加边和删边。

6.2 ETT的做法

这个问题有链操作，ETT是不可能对链进行操作的。LCT中一旦涉及子树标记，也会变得相当复杂。

但是还是有办法将这个问题解决。考虑对一棵树进行欧拉遍历，在ETT中一个点第一次出现之后的连续一段就是从这个点开始不断往第一个儿子移动的路径。这意味着在ETT中将路径表示出来是完全可行的。

那么就可以这么做：结合LCT，将每个点的第一个儿子设为其在LCT中实边连出的唯一的儿子——改变子树顺序在ETT中只不过是改变区间顺序而已。

如果在LCT进行了一次access操作，会依次将一些边变成实边，那么就在ETT中进行相应的修改。对于换根操作，由于LCT和ETT对其都有良好的支持，故而可以解决。考虑这样做的时间复杂度：LCT的时间复杂度为$O(n\log n)$，意味着在ETT中进行的相应操作也是$O(n\log n)$ 次，总时间复杂度为$O(n\log^2 n)$。

接下来考虑查询和修改操作。由于链和子树都对应到了相关的区间，故而可以在ETT中用平衡树进行维护了。

6.3 LCT的扩展

只是利用动态树的话，是没法做子树操作的——而这题的关键就在于子树修改和子树询问，由于有了这个，本题的难度增加了几个级别。但是依旧可以使用LCT做到$O(n\log n)$。

6.3.1 一个简单技巧

本题的修改操作要么是将一些点的权值加上一个数，要么是赋值，故可以将标记设为$a*x+b$的形式，记下a和b的值即可，这样写程序会异常的方便。

6.3.2 一张好看的图

很明显，如果套用传统的动态树的话，问题就在于标记的下传和统计信息的上传上。

考虑一条链：

其中实线是实边，虚线对应于虚边。点是按照深度从高到低。

6.3.3 信息的合并

因为用一个Splay维护实线连起来的链，那么该链里的最大和最小值信息很容易统计。

接下来考虑虚边，由于需要合并子树信息，需用每条虚边连出去的点对应的子树信息[3]来更新实链中的信息。

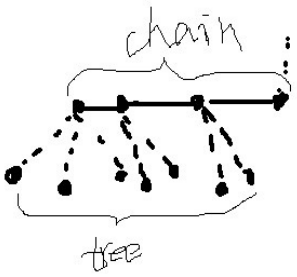
考虑对于一条链需要维护哪些信息：

1. chain: 对于每条链我们统计链内部的点的信息。
2. tree: 所有从这条链上的点连出的虚边对应的子树的信息的和（不包括这条链）。
3. all: 以上两个信息的和，也就是这条链最顶端的点所对应的子树的信息。

注意链是用Splay维护的，也就是说，要在Splay 的每个节点都维护对应的值，实现和普通的动态树没什么区别。

为了看起来更像区间，将树链横放：

[3]因为子树信息可以通过统计链的信息得到，故而之后不区分子树信息和链上的信息



chain的话使用Splay即可维护。如果统计出tree 的话，all只是简单地将其加起来。

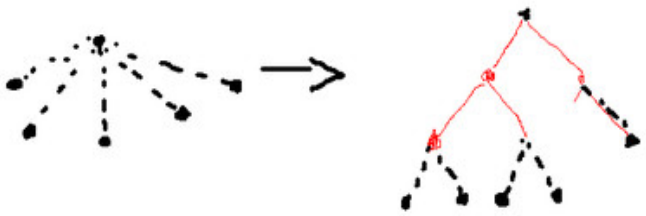
那怎么统计tree呢？

如果对于每个点暴力记下其连出去的所有虚边，就可以直接枚举这些虚边对应的链，用其all来更新该点的tree即可。

但是这个显然是不现实的，因为——太慢了。故而需要在原有的LCT上进行修改：

6.3.4 AAA树

AAA树[4]为此而存在了：对于每个点连出的虚边，同样使用Splay维护这些点，为了写程序方便，这步是通过增加一些内部节点（称之为inner）实现，如图：



其中红色节点就是新增的内部节点。并保证所有的叶子（即黑色节点）就是原树中虚边上的对应的点，内部节点不会超过叶子节点的个数，所以这样做

[4]这只是作者对该结构的称呼，单纯为了叙述方便

是不会改变复杂度的。

显然每条链只会属于一个AAA树，每个点的AAA树包含了这个点连出去的所有虚边。

通过AAA树，可以实现两种操作：

1. add：将一条链的根接到另一条链的某个点上，那么只需将这条链对应的根按照Splay的普通插入插到那个点的AAA树中即可（注意为了保证一定为叶子节点，需要新增内部节点）。

2. del：将一条从虚边连出的链砍掉，使用普通的Splay删除即可。（同样注意这样有可能导致某个黑色儿子个数少于1 的内部节点，则也要将该内部点删除，有一个儿子的话，需要将这个儿子代替自己连到自己的父亲上）

这样，对于实链上的某点的信息进行更新的话，直接使用AAA树的根的信息来更新即可。

6.3.5 AAA树与access操作

接下来考虑在这样的结构下的access——也就是动态树最重要的操作了，有了这个一切动态树的问题基本都解决了。

可以发现，有了add和del两个操作后，对于access操作并没有影响。

在access的每一个循环中，只做了这样几件事：

1. 找到现在的链的父亲（这个直接在该链的AAA树中找到根的父亲即可）。

2. 将一条实边变成虚边，这其实就是add 操作。

3. 将某条虚边变成实边，这其实就是del 操作。

这样在LCT中维护信息的合并也就成为了可能。

6.3.6 标记

怎么实现修改操作，也就是如何实现标记和标记的下传。

很明显有两种标记:

首先是子树标记treeflag，表示这条链对应的子树中的修改（注意不能包含这条链的修改）。其次是链标记chainflag，也就是这条链的修改。

链标记的下传也是简单且常规的。

而treeflag的下传则有两种：

1. 在实链的Splay中（或者是下传到内部节点），直接下传即可，无需修改。
2. 若从AAA树下传到外部节点，需将treeflag拆分成chainflag+treeflag的形式。因为treeflag不包含在链上的修改

6.3.7 实现细节

怎么写呢？看起来AAA树好麻烦啊。

其实不然，只需将LCT中Node的两个儿子改成四个儿子$ch[4]$。

这样只需要传递一个标记type即可决定是考虑在LCT中进行操作还是在AAA树中。

并给内部节点一个inner标记，方便标记。

6.3.8 时间复杂度

在不严谨的分析下，这个做法的复杂度至多不会超过$O(\log^2 n)$。在相关论文9 中证明了这样做的复杂度在均摊意义下单次操作是$O(\log n)$的，虽然有常数为96的因子。但这个做法在实际中也有着较快的运行速度。

6.3.9 名字?

在9中称为Self-Adjusting Top trees。但如果只是看做LCT的一种简单拓展将会更易理解。

6.4 Rake和Compress方法

在11中袁昕颢提到了一种有意思的数据结构：RC tree。

具体来讲，对于一棵树，可以使用两种方式将其划分成更小的子树：Rake节点，代表了以原树中以某个点为根的子树。Compress 节点，代表了原树中一条链上所有点的子树。

利用这两种节点对树进行剖分，可以达到和动态树相类似的效果。加上引入了随机的策略，最终得到期望$O(\log n)$的深度。但在top tree的论文9中提到了这个做法的局限，它要求一棵树的节点个数为常数。若牺牲一定的时间复杂度，也是有可能解决最初的动态树问题。作者并未实现，不敢妄下断言。

7 动态仙人掌

这部分内容是向吕凯风同学致敬。在此会对动态仙人掌问题和相关解法做一定介绍。

7.1 问题描述

定义仙人掌图为每条边最多属于一个简单环的图。

动态仙人掌问题是要求动态地维护一个仙人掌图。由于科技水平不发达，故而目前只限于路径查询和路径修改操作。

对于每条边有一个固定的权值A_i，则路径(u,v)为u到v的所有路径中$\sum A_i$最小的路径。

动态仙人掌这题分为三个难度递增的子问题：

1. 动态仙人掌I：加边和删边，查询最短路径的长度。
2. 动态仙人掌II：在I的基础上对于每条边还有一个固定的权值B，要求查询一条路径上最小的B值。
3. 动态仙人掌III：在II的基础上要求支持对一段路径的B值进行操作。

7.2 一个优秀的算法

最显然的做法就是用LCT维护仙人掌的生成树。可以做到时间复杂度$O(n \log n)$，达到理论最优。

不过该做法的具体实现过于复杂，故而略去对此的讨论。有兴趣的读者请参见吕凯风同学的相关题解。

7.3 Link/cut cactus

为何维护生成树会如此繁琐？因为将一个树上的算法直接套到仙人掌图上本身就无法避免各种复杂的讨论。为了得到一个很好写的做法就不得不跳出树的框架。不妨思考怎么修改Link/cut tree这个数据结构以使之适应于仙人掌图。于是就得到了一个崭新的数据结构Link/cut cactus。

7.4 仙人掌图剖分

仿照LCT，为了得到LCC的做法，必须先考虑如何将仙人掌图分成若干条不相交的链。指定任意一个点为根。

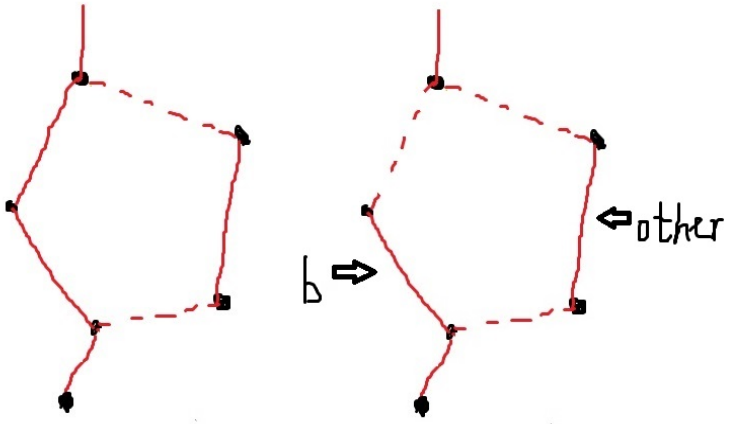
如果不存在环，一切都可以按照LCT进行定义了，否则？

首先必须保证剖分成的链的一定是一条合法的路径（这条链为两个端点的最短路）。如果将环看成LCT中的一个点，还要保证对于一个环，至多只会往下（远离根的方向）连出一条边。

其次，虽然一个点可以属于任意多的环，但只能属于一条链。

将链中的边称为实边，不在链中的边称为虚边。

令一个环中离根最近的点a为这个环的根，设点b有往下连出的边。则可能存在如下两种剖分方式：



其中实线和虚线表示剖分中实边和虚边。

注意到，除去根，一个环最多分成两条链，且仅有一部分的一个端点会存在向下的实边，称这部分为b链，另一部分为other链。

两者的不同点在于根与环中其它点是否相连。可以看出，一个点可能成为许多环的根，但至多在一个环的b链或者other链中。这意味着可以直接维护所有的链。

7.5 access

下面考虑access，对于动态树来说，只要能够实现access操作问题就基本解决了。

首先考虑其意义：将一个点到根的最短路径连成一条链。

假设在access中当前的点为u，查找u在*LCC*上的往前连的边（可以使用链和最短路径定义）。如果这条边不在环中，则做法类似于LCT。

否则，找到这条边所在的环。可以肯定u不是这个环的根（设为a）。故而可以实现操作：将b链和other链连接起来，从u处断开，使得点u 位于较短的那条链上，得到新的b和other链，而且点u刚好为b的端点。将这个操作称为Expand

将u往下的链与新的b链连接即完成了对点*u*的操作。

注意点a：如果a与本来的b链相连，需在最后将a与新的b链连接起来。否则，先对点a进行了Expand操作，再与新的b链相连。

7.6 换根

很幸运的是，换根操作非常简单：对新的根节点进行access操作，得到路径，将其翻转。

注意，LCC要求对每个环维护相应的a,b,other。经过翻转之后会发生相应变化。解决方法很简单：每次访问到一个环，在路径中查询a,b的顺序，一旦顺序不同，则将环中相应的链进行翻转，相应的值对调。

7.7 其它操作

有了access和换根操作，其它操作就非常简单了。

7.8 一些小细节

因为此题的权值都在边上，故而要对每个点维护链中与之相连的两条边，使得在翻转操作后依然可以得到正确的边的集合。

此题要求判断最短路径是否唯一，解决方法是对于每个环记录b链和other链长度是否相等，从而可以对相应的路径进行标记。

7.9 时间复杂度

由于LCC的中实边和虚边的切换与LCT相类似，总次数不会超过$n \log n$，考虑到Splay，时间复杂度为$O(n \log^2 n)$[5]。经实测，效率也是挺不错的。

7.10 一些推广方向

LCC是和LCT一样非常灵活的数据结构，可做如下探讨：

1. 是否可以可持久化。
2. 是否可以将self adjusting top tree和LCC结合，使得能够解决子仙人掌的操作和查询。
3. 是否可以将dfs序推广到仙人掌图上。
4. 如何使用LCC来优化仙人掌图上的统计和最优化问题。

欢迎有兴趣的同学继续思考相关的问题。

后记

本文对信息学竞赛中涉及到的动态树问题进行了简单的介绍。

选择这个简单的题目，原因可能是因为个人对这方面的问题比较感兴趣。近两年来动态树成为各种模拟比赛中的热门题目，题目难度越来也大，类型越来越多。故而有了对这类问题进行总结的想法。

由于时间仓促，以及作者能力有限，很多问题都没有涉及。不过也希望这篇论文能够帮助到将要或正在学习动态树的OIer们，这样的话，本文也就能有所意义了。

[5]怀疑时间复杂度为$O(n \log n)$，但是无法证明

感谢

感谢廖晓刚老师的在学习生活上的关心和照顾。

感谢朱全民老师和屈运华老师的教导。

感谢国家集训队教练的辛勤付出。

感谢CCF提供的平台和机会。

感谢匡正非，刘研绎，毛啸，彭雨翔，杨定澄，刘剑成，谭博文，杨卓林，苏雨峰等同学提供的意见和帮助。

感谢陈立杰，我从他那儿学到了top tree及其它各种各样有趣的算法。

感谢吕凯风在动态仙人掌问题上做出的开创性的研究。

感谢唐翔昊，魏子昆，陈思琪，仇知，徐诗雨，黄骏翔，王子骏等同学陪我度过了快乐的时光。

参考文献

[1] 刘汝佳, 黄亮,《算法艺术与信息学竞赛》, 清华大学出版社。

[2] 刘汝佳,《算法竞赛入门经典》, 清华大学出版社。

[3] 黄志翱,《浅谈算法竞赛中的简单构造题》

[4] 杨哲,《SPOJ375 QTREE解法的一些研究》, 2007年国家集训队作业。

[5] 何朴藩,《HNOI 2010 解题报告》, 2011年国家集训队作业。

[6] 陈立杰,《重量平衡树和后缀平衡树在信息学奥赛中的应用》, 2013年国家集训队论文。

[7] 6.851: Advanced Data Structures L20

[8] 匡正非, 黄志翱,《两个冷门图论算法》, 2014年信息学奥林匹克冬令营。

[9] Robert E. Tarjan, Renato F. Werneck, Self-Adjusting Top Trees.

[10] 陈首元, 维护森林连通性——动态树

[11] 袁昕颢, Dynamic Trees Problem, and its applications.

[12] 吕凯风，《动态仙人掌系列题解》