RMQ 与 LCA 问题的解决及应用

长沙市长郡中学 郭华阳

目录

【正文】.....1
RMQ 与 LCA 问题的提出.....1
一. LCA 问题的提出.....1
RMQ 与 LCA 问题的可转化性.....2
一. RMQ 问题转化为等规模的 LCA 问题.....2
二. LCA 问题转化为等规模的 RMQ 问题.....3
三. RMQ 与 LCA 问题的可转化性的重要意义.....4
RMQ 与 LCA 问题的解决.....4
一. 解决 RMQ 与 LCA 问题的算法概述.....4
二. 解决 RMQ 与 LCA 的朴素算法.....4
三. 解决 RMQ 问题的 Spare Table 算法.....5
四. 解决 LCA 问题的 Tarjan 算法.....6
五. 解决±1RMQ 的 BF 算法.....7
六. 各种算法的联系.....8
RMQ 与 LCA 问题在信息学竞赛中的应用.....8
例一. 水管局长（Winter Camp 2006）.....8
例二. Query on a Tree(SPOJ 375，QTREE).....11
总结.....12
【参考文献】.....13
【感谢】.....13

【正文】

RMQ 与 LCA 问题的提出

RMQ（Range Minimal Query）问题即区间内最小值询问问题。

让我们考虑一个线性序列 A = （A_1, A_2, … A_n），我们往往需要处理形如：A[l, r] = (A_l, A_{l+1}, … A_{r-1}, A_r)中的最小值是多少？这一类问题我们称之为区间最小值询问问题，并且以 RMQ(A, l, r)表示询问 A[l, r]中的最小值。

以一个简单的实例说明，对于线性序列 A = (3, 2, 1, 5, 4)，我们有：

RMQ(A, 1, 3) = A_3

RMQ(A，2，5) = A_3

RMQ(A，4，5) = A_5

一. LCA 问题的提出

LCA（Least Common Ancestors）即最近公共祖先问题。

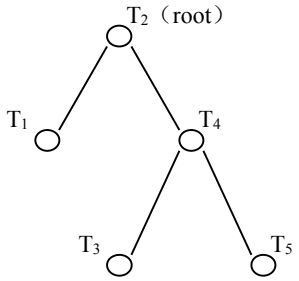
一个出现在有根树中间的常见问题是：在有根树 T 中询问一个距离根节点最远的结点 x（即深度最大的结点）使得 x 同时为结点 T_u、T_v的祖先，我们称 x 为 u、v 的最近公共祖先。这种对于公共祖先的询问我们称之为公共祖先问题，并且以 LCA(T, u, v)表示询问 T 中结点 T_u、T_v的最近公共祖先。

考虑一个简单的有根树 T（如图）：我们有：

LCA(T, 1, 3) = T_2

LCA(T, 3, 4) = T_4

LCA(T，3，5) = T_4



RMQ 与 LCA 问题的可转化性

RMQ 问题与 LCA 问题看似两个毫不相关的问题，但接下来我们将会看到，两个问题是可以互相转化并且不会改变问题规模的。这正是这两个问题一同出现在我们视线里的原因之。（脚注：简便起见，文中所讨论的线性序列 A 中任意两元素均不相等）

一. RMQ 问题转化为等规模的 LCA 问题

为了更加清楚地解释 RMQ 向 LCA 的转化，我们先介绍优先级树以及笛卡儿树的概念：

优先级树是一棵有根树，每个结点有一个优先级 Priority，并且满足任意结点祖先的优先级不大于该结点的优先级，不难发现，根节点具有优先级树中优先级最大的结点。右图给出一棵优先级树。

笛卡儿树是一棵有根二叉树，每个结点有两个关键字：Priority 和 Value。对于 Priority 而言，笛卡儿树是一棵优先级树；对于 Value 而言，笛卡儿树是一棵排序二叉树。

将 RMQ 转化为 LCA 的关键思想在于笛卡儿树的结构：

引理一：

对于线性序列|A| = N，如果存在有根树|T| = N 使得 T 的中序遍历结果为（T_1，T_2，…T_N）。T_k加权 A_k，并且以 A_k为 Priority 树 T 是一棵优先级树。那么必然有 RMQ(A, l, r) = LCA(T, l, r)

证明：

采取反证法，假设 RMQ(A, l, r) = Ap 且 LCA(T, l, r) = Tq 且 p ≠ q。考察 T 中以 Tq 为根的子树 NT，则必然有 T[l, r]∈ NT，于是 Tp ∈ NT。但是根据假设 Ap < Aq，也就是说 Tq 的优先级比 Tp 高，与优先级树的性质矛盾！

根据引理一，我们只需要找到满足条件的 T 即可。不难发现 T 即一棵以 A_i为 Priority、i 为 Value 的笛卡儿树，对于给定线性序列一定存在对应的 T。以下我们给出一种朴素的递归构造方法：

朴素的笛卡儿树的建立算法：

1） 找到 A 中的最小元素 A_k，以 A_k作为 T_k的优先级；

2） 将 A[1, k-1]建立笛卡儿树，作为 T_k的左子树；

3） 将 A[k+1, N]建立笛卡儿树，作为 T_k的右子树；

不难证明这样建立起来的有根树 T 一定是满足条件的，这也就证明了 RMQ 问题一定可以

转化成为等规模的 LCA 问题。

但是朴素算法建立笛卡儿树的时间复杂度过高。如果采取枚举法寻找最小元素 A_k 算法总共需要 $O(N_2)$ 的时间，即使采用数据结构进行优化也需要 $O(N\log_2 N)$ 的时间。

事实上，存在一种 O(N)的时间内建立笛卡儿树的算法：

假设 T(k)为基于 A[1, k]建立出的笛卡儿树，算法由 T(k-1)计算 T(k)。

我们将 T_k 插入为 T_{k-1} 的右孩子得到 T0。事实上，若 $A_k > A_{k-1}$，则 T0 即为 T(k)了。否则的话，不难发现 T0 中只有 T_k 不满足笛卡儿树的条件：任意结点的优先级低于祖先。

引理二：

若 T0 中只存在一个结点不满足笛卡树的条件，对该结点进行一次旋转操作使其深度减小，此时 T0 中至多只存在该结点不满足笛卡树的条件

证明：略

根据引理二，对 T_k 不断进行旋转操作直到 T_k 的优先级低于其父亲结点的优先级或者 T_k 成为根节点，此时 T0 成为一棵笛卡儿树，即为 T(k)。

据此，我们可以得到该算法流程：

建立笛卡儿树的线性算法：

1) T(0) = nil;
2) 如果以获得 T(k-1)，将 T_k 插入到 T_{k-1} 的右孩子，不断旋转 T_k 直到 T_k 满足笛卡儿树的性质；
3) 输出 T(N)

我们来计算一下该算法的时间复杂度：

设置势函数 $\Phi(k)$ 表示 T_k 在 T(k)中的深度，则算法的总耗时为 $\sum|\Phi(k-1)-\Phi(k)+1|$。因为 $0 \leqslant \Phi(k) \leqslant \Phi(k-1)+1$，所以 $\sum|\Phi(k-1)-\Phi(k)+1| = \sum(\Phi(k-1)-\Phi(k)+1) = \Phi(N)+N = O(N)$。如此我们证明了该算法的时间复杂度为 O(N)。

综上，RMQ 问题可以在 O(N)的时间内转化为等规模的 LCA 问题。

二. LCA 问题转化为等规模的 RMQ 问题

LCA 问题向 RMQ 问题的转化与其逆转化有所不同，因为 LCA 问题可以建立在多叉树上而不仅仅是二叉树上，所以我们不能单纯的通过笛卡儿树来进行转化。为了解决这个问题，我们提出欧拉序列的概念：

对有根树 T 进行 DFS（深度优先遍历），无论递归还是回溯将到达的每一个结点记录下来，如此将会得到一个长度为 2N－1 的序列，我们称之为树 T 的欧拉序列 F（类似欧拉回路）。

举例说明：

我们把 T_k 在欧拉序列中第一次出现的位置记录为 pos(k)。

引理三：

对于 T 的欧拉序列 F，有 LCA(T, u, v) = RMQ(F, pos(u), pos(v))

证明：

以下两点显而易见

1) 根据 DFS 的性质，在从 u 到 v 的过程中一定会经过 LCA(T, u, v)；
2) 根据 DFS 的性质，在从 u 到 v 的过程中不会经过 LCA(T, u, v)的祖先；

据此可以得到 LCA(T, u, v) = RMQ(F, pos(u), pos(v))

对树 T 进行 DFS 得到欧拉序列的时间复杂度是 O(N)，且欧拉序列的长度为 2N-1 = O(N)，

所以，根据引理三 LCA 问题可以在 O(N)的时间内转化为等规模的 RMQ 问题。

三． RMQ 与 LCA 问题的可转化性的重要意义

RMQ 与 LCA 问题的可转化性使得两种算法可以有机结合，同时也使得建立在两种问题模型上的算法可以互相交融、促进，这一点我们将在下一届里发现。另外，在解决实际问题的时，也为我们提供了一种新的思路。

RMQ 与 LCA 问题的解决

一． 解决 RMQ 与 LCA 问题的算法概述

在讨论关于 RMQ 与 LCA 的算法之前，我们先了解一些对于这些算法性能的描述。此类算法可以按照在线与离线被分为两大类：

在线算法：

通过预处理，每读入一次询问就可以直接给出回答。我们用 O(f(N))-O(g(N))来描述一个在线算法的时间复杂度，其意义为：在 f(N)的时间内完成预处理，此后对于每一个询问使用 g(N)的时间回答

离线算法：

一次性读入所有询问，在一次算法的执行过程中得到全部的答案，然后一起作答。我们用 O(f(N))描述离线算法的时间复杂度，其意义为：在 f(N)的时间内完成所有询问。

经过研究人员的努力，RMQ 与 LCA 问题已经有了比较完善的解决方法，本文主要介绍的几种算法有：

1) 朴素算法；
2) $O(N\log_2 N) - O(1)$的 ST(Spare Table)算法；
3) $O(N\alpha(N) + Q)$的 Tarjan 算法；
4) $O(N) - O(1)$的 BF(±1RMQ)算法；

需要注意的是，以上算法中大多数不能完成修改 A 或者 T 的操作，所以这一操作不在我们的讨论范围（实际上可以通过平衡树来实现）。

二． 解决 RMQ 与 LCA 的朴素算法

建立在 RMQ 问题上的询问只有 $O(N^2)$种，且 RMQ(1, r)<short for RMQ(A,1,r)>可以通过 DP 得到，据此我们可以设计离线算法：

> 解决 RMQ 问题的 $O(N^2)$-O(1)算法 A：
> 1) RMQ(k, k) = A_k；
> 2) 根据 RMQ(1, r) = min{RMQ(1, r-1), A_r}推出所有状态的解；
> 3) 对于询问 RMQ(A, 1, r)通过 RMQ(1, r)使用 O(1)的时间作答

RMQ 的另一种在线算法可以通过线段树等数据结构来实现，这种对于数据结构的利用本文不再赘述，可以参考前辈的论文，其时间复杂度为 $O(N)-O(\log_2 N)$。

在有根树 T 中，记录 Depth(u)为结点 u 到根节点的距离，也就是它的深度。

考虑 LCA(T, u, v)，不妨设 Depth(u) >= Depth(v)：

1） 若 u = v，那么显然有 LCA(T, u, v) = u；

2） 否则有 LCA(T, u, v) = LCA(T, father(u), v)；

据此，我们可以设计一个不需要预处理在线回答询问的算法：

解决 LCA 问题的 O(1)-O(N)算法 B：

$$LCA(T,u,v)=\begin{cases}u & u=v\\ LCA(T,father(u),v) & u\neq v\end{cases}$$

同 RMQ 问题相同，建立在 LCA 问题上的询问不超过 $O(N^2)$种，我们同样可以预先处理所有的询问，然后 O(1)的作答。略有不同的是，LCA(u,v)<short for LCA(T,u,v)>的 DP 需要自顶向下：

解决 LCA 问题的 $O(N^2)$-O(1)算法 C：

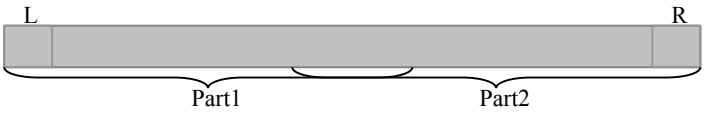
1） LCA(u,u) = u；

2） 自顶向下取结点 u，LCA(u, v) = LCA(father(u), v)

这里我们之所以仔细描述算法 A、B、C，是因为这些基础算法是我们得到更优算法的基础。

三． 解决 RMQ 问题的 Spare Table 算法

ST(Spare Table)算法是在 $O(N\log_2 N)$-O(1)内完成 RMQ 问题的在线算法。算法充分利用了以下思想：



如上图所示，考虑 RMQ(A,l,r)，如果我们已经获得了 RMQ(A, part1)与 RMQ(A，part2)并且 part1 与 part2 的并区间恰好为[l, r]那么有：

RMQ(A,l,r)= min{RMQ(A, part1), RMQ(A，part2)}

为了获得更快的速度与更少的状态，我们只需处理长度为 2 的幂的区间的 RMQ，利用倍增思想加速计算。具体来说，记录 f(k,d) = RMQ(A, k, $k+2^d-1$)，这样的状态总共有 $O(N\log_2 N)$种，根据动态规划方程：f(k, d) = min{f(k, d－1), $f(k+2^{d-1}, d-1)$}我们可以在 $O(N\log_2 N)$的时间内计算出所有 f(k, d)的结果。

对于询问 RMQ(A, l, r)，我们可以找到两段极大的长度为 2 的幂的区间覆盖 A[l,r]，具体的说，取 $d=\lfloor \log_2(r-l+1) \rfloor$，则区间[l, $l+2^d-1$]与区间[$r-2^d+1$,r]覆盖了区间[l,r]。

我们给出 ST 算法的流程：

解决 RMQ 问题的 ST 算法：

1） 根据 $F(k,d)=\begin{cases}A_k & d=0\\ Min\{F(k,d-1),F(k+2^{d-1},d-1)\} & d>0\end{cases}$ 预处理

所有长度为 2 的幂的区间的最小值；

2） 对于询问 RMQ(A，l，r)，取 d = $\lfloor \log_2(r-l+1) \rfloor$，有：

RMQ(A，l，r) = Min{F(l, d),F(r － 2^d + 1, d)}

不难证明 ST 算法的时间复杂度为 $O(N\log_2 N)$-O(1)。

四. 解决 LCA 问题的 Tarjan 算法

Tarjan 算法是 O(Nα(N) + Q)解决 LCA 问题的离线算法，其中α(N)在一般情况下是一个不大于 5 的实数。

Tarjan 算法基于以下事实实现：

有根树 T 中，记录 V(k)表示某一深度为 k 的结点，V(k-1)为 V(k)的父亲结点，V(k-2)为 V(k-1)的父亲，……，V(0)为根结点。

考察一组询问(v(k)，u)：

1) 若 u 在子树 v(k)中，那么 LCA(T, v(k), u) = v(k)
2) 若 u 在子树 v(k-1)中而不在 v(k)中，那么 LCA(T, v(k), u) = v(k-1)
3) 若 u 在子树 v(k-2)中而不在 v(k-1)中，那么 LCA(T, v(k), u) = v(k-2)

……

k) 若 u 在子树 v(1)中而不在 v(2)中，那么 LCA(T, v(k), u) = v(1)；

k+1) 否则 LCA(T, v(k), u) = v(0)；

我们把这样一条从 v(k)到 v(0)的路径称之为 v(k)的活跃路径，记为 P(v(k))。

引理四：

LCA(T, u, v) = P(u)上距离 v 最近的结点

证明：

由以上事实可知

利用引理四，Tarjan 算法利用并查集在一次 DFS 中完成所有询问。

对于每一个组询问 LCA(T, u, v)，假设 u 在 v 之后被遍历到，那么我们在访问 u 时处理询问 LCA(T, u, v)。对于 DFS 过程中正在访问的结点 u_0，称活跃路径 $P(u_0)$为当前活跃路径。对于每一个已经遍历过的结点 w，我们记录 H(w)表示当前活跃路径上距离结点 w 最近的一个结点。根据引理四，有：$LCA(T, u_0, w) = H(w)$。

那么我们可以设计如下的算法流程：

Tarjan 算法流程一：

TarjanDFS(u)

1) H(u) ← u，将 u 标记为已遍历过；
2) 对于每一组与 u 有关的询问 LCA(T,u,v)，若 v 已经遍历过，则 LCA(T,u,v) ← H(v)；
3) for v∈Son(u)
   a) TarjanDFS(u)；
   b) 维护 H；

现在仍然遗留下来一个问题，如何对 H 进行维护，除去这一步以外，其他步骤都是线性时间内完成的，为了得到较优的时间内完成，我们要使得这一套的时间复杂度降低。

H 的维护有两个步骤：

1) 新遍历到一个结点 u 时，令 H(u) = u；
2) 访问完 u 的子树 v 后回溯回结点 u，此时对于任意 w∈子树 v 均有 H(w) = v，我们需要将其更新为 H(w) = u；

Tarjan 巧妙的利用了并查集的特点完成以上两个步骤：

1) 新遍历到一个结点 u 时，新建集合 set(u) = {u}，代表元为 u；

2) 访问完 u 的子树 v 后回溯回结点 u，另 set(u) = Merge{set(u), set(v)}，代表元为 u；

引理四：
H(T_u) = T_u所在 set 的代表元。
证明：
通过并查集的性质可知。

这样我们就可以在 O(Nα(N))的时间内支持 H 的维护了。
综合以上所述，我们得到 Tarjan 算法完整的算法流程：

Tarjan 算法流程二：
TarjanDFS(u)
1) Set(u) ← {u}、代表元为 u，将 u 标记为已遍历过；
2) 对于每一组与 u 有关的询问 LCA(T, u, v)，若 v 已经遍历过，则 LCA(T, u, v) ← v 所在集合代表元；
3) for v∈Son(u)
   a) TarjanDFS(u)；
   b) Merge{Set(u), Set(v)}，代表元为 u；

不难证明，Tarjan 算法的时间复杂度为 O(Nα(N) + Q)。

五. 解决±1RMQ 的 BF 算法

在前文中我们提到过，LCA 问题可以转化为欧拉序列的 RMQ 问题。其特殊之处在于，欧拉序列中任意两个相邻的元素深度之差恰好为±1！我们把建立在这样一种特殊序列上的 RMQ 问题称为±1RMQ 问题，与此区别的，其他情况我们称之为一般 RMQ 问题。

在±1RMQ 问题上，Michael A. Bender 和 Martín Farach-Colton 提出了 O(N)-O(1)的在线算法。

算法的中心思想在于分块，我们把元素依次以块长 L = 2N / $\log_2$N 分成 M = N / L 块。记第 i 块为 $Block_i$={$A_{(i-1)L+1}$…A_{iL}}，我们记录其中元素的最小值为 $BlockMin_i$，记 Blocks(1…M) = {$BlockMin_1$, $BlockMin_2$, …, $BlockMin_M$}。

根据分块思想，对于询问 RMQ(A, l, r)，设 A_l∈$Block_u$、A_r∈$Block_v$:

1) 若 u = v，那么 RMQ(A, l, r) = RMQ($Block_u$, l - uL + L, r - uL + L)；
2) 若 u < v，那么 RMQ(A, l, r) = Min{RMQ(Blocks, u + 1, v - 1), RMQ($Block_u$, l - uL + L, L), RMQ($Block_v$, 1, r - uL + L)}；

我们称 RMQ(Blocks, 1, r)为 Block-RMQ，RMQ($Block_u$, l, r)为 In-RMQ。接下来我们将看到，Block-RMQ 和 In-RMQ 都是可以在 O(N)-O(1)的时间内解决的。

Block-RMQ 的问题规模只有 M = N / L，我们应该充分利用这一性质，使用 ST 算法予以解决。根据前文所述，ST 算法解决 Block-RMQ 的时间复杂度为 O(M$\log_2$M)-O(1)，因为 M$\log_2$M = (2N / $\log_2$N) * $\log_2$(2N / $\log_2$N) < O(N)，所以 Block-RMQ 的时间复杂度不大于 O(N)-O(1)。

关键是 In-RMQ 问题如何有效地解决，这里我们必须用到±1RMQ 相邻两数相差恰好为±1 的特点了。

引理五：
考察 $Block_i$ 和 $Block_j$，若对于任意 1 <= k < L 均有 $Block_i$(k+1) - $Block_i$(k) = $Block_j$(k+1) - $Block_j$(k)，那么下标上一定有 RMQ($Block_i$, l, r) = RMQ($Block_j$, l, r)。
证明：略

根据引理五我们知道，至多只有 $2^{L-1}=O(\sqrt{n})$ 种本质不同的 Block，而对于每一种 Block，至多只有 $O(L^2)$ 种询问，根据解决 RMQ 的朴素算法 A，我们可以得到 $O(L^2)$-O(1)解决一种 Block 的在线算法，由于有 $O(\sqrt{n})$ 种 Block，所以我们可以得到 $O(\sqrt{n}L^2)$-O(1)的算法。由于 $O(\sqrt{n}L^2)=O(\sqrt{N}\log_2^2 N)$ 在一般意义下认为小于 O(N)，所以 In-RMQ 的时间复杂度不大于 O(N)-O(1)。

综上所述，由于 Block-RMQ 和 In-RMQ 都可以在 O(N)-O(1)内解决，所以±1RMQ 问题也可以在 O(N)-O(1)内解决。我们现在给出±1RMQ 完整的算法流程：

O(N)-O(1)的±1RMQ 算法流程：
1) 以 L = 2N / $\log_2$N 将 A 分成 M = N / L 块；
2) 对 BlockMin 进行 RMQ 预处理；
3) 对每种本质不同的 Block 进行算法 A 预处理；
4) 对于 RMQ(A, l, r)分 Block-RMQ 和 In-RMQ 进行回答；

六. 各种算法的联系

正如第二部分所述，RMQ 问题与 LCA 问题可以在线性时间内等价转化，所以任意解决 RMQ 的算法可以在相同的时间内解决 LCA 问题；任意解决 LCA 问题的算法，可以在相同的时间复杂度内解决 RMQ 问题。

RMQ 与 LCA 问题在信息学竞赛中的应用

RMQ 与 LCA 问题在信息学竞赛中主要是以一种经典算法和解题思想的形式出现。我们以两道例题略加体会：

例一. 水管局长（Winter Camp 2006）

[题目简述]

给定带权无向图 G，我们定义 G 中一条路径的花费为路径中权值最大的边的权值，特别的，我们称这条边为路径的关键边。两点之间的最短路定义为连接两点的所有路径中费用最小的一条。

给出 G 上的 Q 的操作要求按顺序完成，操作有两种形式：

1） 拆除无向边(u, v)；

2） 计算 u，v 之间的最短路花费；

任务是在尽量短的时间内完成所有操作。

[数据范围]

结点数 N ≤ 1000；

边数 M ≤ 100000；

操作数 Q ≤ 100000；

删边操作 ≤ 5000；

[分析]

本题最朴素的算法在周戈林同学 2006WC 的 Thesis 中已经提到过，本文就不再赘述了。朴素算法的时间复杂度大致为 $O(N^2Q)$。

首先我们有必要分析的就是此题难以解决的原因：

1) 边数过多，边数过多会增大任何操作的时间花销；

2) 完成询问的时间复杂度过高，就数据范围而言，完成每个询问至多只能使用 $O(\log_2 N)$ 的时间；

解题关键便是：对症下药，削减冗余。

如何才能削减使用的边数？一种常用的思想是我们联想到了树结构，的确，树结构的边数只有 O(N)级别，并且仍然维护了图 G 的连通性。我们猜测，存在一种树结构，使得任意询问均可在这个树结构上找到最优解。

引理六：

任意一组询问均可在图 G 的最小生成森林 TG 中找到最优解。

证明：

对于路径 P，记 Φ(P)表示路径 P 中不在 T 中的边数；

1） 若 Φ(P) = 0，则 P 在 T 上；

2） 若 Φ(P) > 0，则存在一条边(u, v)∈P - T

考察这条边(u, v)：

a) 根据最小生成森林的性质我们知道，存在一条只有树边构成的路径连接(u, v)两点，并且该路径 P0 的花费不大于(u, v)的花费；

b) 将 P 中边(u, v)替换为该路径，P 的花费不变大，且 Φ(P) 减小；

由于 Φ(P) ≥ 0，所以可以在有限次替换后将 P 变成一条 T 中的路径而费用不增。

得证。

根据引理六可知只需要保存 O(N)的边数就可以了。但是题目设计的非常巧妙，题目中加入了删边操作，这使得我们需要重新建立最小生成树。而建立最小生成树是 $O(M\log_2 M)$ 或者是 $O(N^2)$ 的，我们需要优化。换一个方向思考：

引理七：

如果无向边 e0 当前不在最小生成森林中，新加入一条无向边，e0 仍然不在最小生成森林中。

证明：

略

根据引理七，我们可以将操作执行顺序倒转，从后向前执行，如此删边操作就变成了添边操作。每添加一条无向边，如果采用 Kruscal 算法，我们只需要新边添加到原先的树边组成的按权值有序表中，在执行 Kruscal 算法即可，时间复杂度为 O(Nα(N))。特别的，我们需要生成结束时刻的最小生成树，时间复杂度为 $O(M\log_2 M)$

我们解决了第一个问题，即边数已经下降到 O(N)级别，但是完成询问的时间复杂度依然需要优化。虽然最小生成森林中两点路径是唯一的，但是如果采取 DFS 来完成，复杂度为 O(N)仍然太高了。

我们需要一些联想思维，从最小生成树的建立与询问的完成之间是否能找出一些思路呢？

在 Kruscal 算法中，我们考虑最小生成森林建立过程中图 G 的连通性问题。当我们加入有效树边 E_i 时，实际上是将两个连通块合并为一个连通块。

引理八：

若有效树边 E_i 的添加使得连通块 A 和 B 合并为一个连通块。那么对于任意顶点 u∈A、v∈B 均有 MCP(u, v)的关键边为 E_i，这里 MCP = Minimum Cost Path，即最小花费路径。

证明：

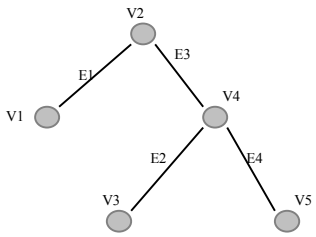
1) 在 E_i 之前添加的有效树边的费用都不大于 E_i，并且 E_i 添加后使得 u、v 连通，所以存在路径经过 E_i 连接 u、v 使得 E_i 为关键边；
2) 根据最小生成森林中 u、v 的路径是唯一的可知，E_i 是唯一的关键边；

得证。

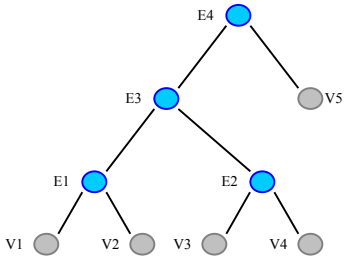
根据引理八，我们知道，两个顶点 u、v 之间的关键边，实际上就是一条**最早**添加的树边，使得在其添加之后 u、v 连通。

为了解决“最早”这个问题，我们引入 Kruscal 顺序生成树的概念（简称为顺序树）。在 Kruscal 算法中，我们给每一个连通块设置一个代表元，初始时刻每个顶点为其所在连通块的代表元。当有效树边 E_i，加入时，假设将连通块 A 与连通块 B 连接起来形成新的连通块 C，那么我们建立一个虚结点代表 E_i，并将其作为连通块 C 的代表元。如此，我们可以构建出一个有 N + MT(有效树边)的生成森林。

我们以一个例子来说明：



最小生成树



相应的顺序树

在顺序树上分析可知，“最早”使两个顶点连通的有效树边就是顺序树中两个顶点的最近公共祖先！

于是我们利用顺序树把询问转化为 LCA 问题了。

两次添边操作中间所有的询问可以看作是离线询问，利用已有的 Tarjan 算法在 O(Nα(N)+Q)的时间内完成。同时，顺序树的生成和最小生成森林的生成是同步进行的。按照前文所述可以很快地解决。

这样第二个问题，即回答询问花费的时间也下降了。我们归纳一下算法流程：

水管局长算法流程：

1) 生成结束时的最小生成树和顺序树，时间复杂度为 $O(M\log_2 M + M\alpha(M))$；
2) 从后往前完成操作：对于添边操作，重新生成最小生成树和顺序树；对于连续的询问操作，将其作为离线 LCA 询问在顺序树上处理；这一步的时间复杂度为 $O(Q + 5000N\alpha(N))$
3) 输出答案

该算法的时间复杂度为 $O(M\log_2 M + Q + 5000N\alpha(N))$，可以很好的解决本题了。

例二. Query on a Tree(SPOJ 375，QTREE)

[题目简述]

给定一棵 N 个结点的树 T，我们将树边按照 1 至 N-1 编号。你需要设计一种算法支持一下两种操作：

1） 将边 i 的费用更改为 ti；

2） 询问 T 中结点 a 到结点 b 路径上花费最大的边。

[数据范围]

N <= 10000；

操作数 <= 100000；

[分析]

无根问题有根化，任取一个结点为根使得树 T 变为一个有根树。

对于操作 2），询问 T 中结点 a 到结点 b 路径上花费最大的边，我们可以将其视为由两部分构成：a 到 LCA(T, a, b)的路径上花费最大的边；b 到 LCA(T, a, b)的路径上花费最大的边。这样我们便可以将操作放到最近公共祖先的问题模型上来了。

根据解决 LCA 问题的算法 B，我们可以设计一个类似的过程来完成操作二：

QTREE 朴素算法：

我们不妨假设 Depth(u) >= Depth(v)，则：

$$Query(u,v)=\begin{cases} nil & u=v \\ Max\{Query(father_u,v),Cost(u,father_u)\} & u\neq v \end{cases}$$

递归求解

通过递归我们可以在 O（N）的时间内解决一次询问，但是考虑到题目给出的数据范围，这样的时间复杂度是无法承受的。纵观前文所提到的经典算法，也并没有哪一种能够很好的解决本题。那么，我们需要创造一种新的 LCA 算法以适应本题的需求！

朴素算法的优势在于：修改边权是 O(1)的；缺陷在于：询问是 O(N)的；

使用经典算法也许可以解决完成询问复杂度过高的问题，但是不支持修改边操作！

新的 LCA 算法需要在两者之间找到平衡点。让完成询问时间和修改边权时间平衡，让我们想到了 $\sqrt{n}$，这让我们联想到了分块思想，经典分块链表的时间复杂度恰好为 O($\sqrt{n}$)。

假设我们可以将树 T 划分为 $\sqrt{n}$ 个连通块，每个连通块恰好有 $\sqrt{n}$ 个结点，我们纪录第 i 个连通块为 T(i)，显然 T(i)是一棵有根树。纪录 KeyEdge(k)表示结点 k 到其所在分块的根结点的路径上权值最大的边。根据：

$$KeyEdge(u)=\begin{cases} nil & u=root \\ Max\{KeyEdge(father_u),Cost(father_u,u)\} & else \end{cases} \quad (*)$$

我们可以在 O(N)的时间内预处理所有的 KeyEdge(k)。

考察 Query(u, v)，设 u∈T(i)、v∈T(j)：

1） 若 i = j，我们只需在 T(i)中执行朴素算法，时间复杂度控制在 O($\sqrt{n}$)；

2） 若 i ≠j，并且至少有一个结点为其所在分块的根结点，不妨为 u，显然有：

$Query(u,v)=Max\{Query(father_u,v),Cost(u,father_u)\}$；

3） 否则……；

引理九：

对于 3)，不妨设 T(i)根结点的深度不小于 T(j)根结点的深度，那么：p =LCA(T, u, v)的深度一定小于 T(i)的根结点 q；

证明：

如若不然，p 一定在子树 q 内。而 v 一定在子树 p 内，于是 v 在子树 q 内。T(j)的根结点深度不大于 q，而 v 在子树 q 内，因为 T(j)是连通的，于是有：q∈T(j)，矛盾！

根据引理九，情况 3)时，有：$Query(u,v) = Max\{Query(q,v), KeyEdge(u)\}$

情况 1)只出现一次，情况二之多出现$\sqrt{n}$次，情况三之多出现$\sqrt{n}$次，所以完成一次询问的时间复杂度为 O($\sqrt{n}$)。

分块思想对于修改边的操作同样优秀：

1） 若修改的边不属于任何一个连通块，即该树边连接两个不同的连通块，那么不用作任何操作；

2） 否则由(*)我们可以重新计算其所在连通块的 KeyEdge，时间复杂度为 O($\sqrt{n}$)。

这样我们可以在 O($\sqrt{n}$)完成修改边权操作。

最重要的问题是，理想的分块情况是达不到的，我们应该怎样分块才能达到比较理想的情况呢？

为了保证复杂度，分块需要满足两个要求：

a) 每个块中的元素至多有$\sqrt{n}$个；

b) 每个结点到 T 的根结点的路径上至多经过$\sqrt{n}$个块；

事实上，这一步分块是很容易办到的，在一次 DFS 中就可以确定所有的块划分：

DFS(u)

1) 若 u 不属于任意一块，将其新建为单独的一块；

2) 对于 u 的每个孩子结点 v：

a) 若 u 所在块的结点数小于$\sqrt{n}$，那么将 v 加入该块；

b) DFS(v)；

不难证明这样一种“简单”的划分是满足条件的，具体的证明就留给读者了。

于是，我们得到了一种在 O(Q$\sqrt{n}$ + N$\sqrt{n}$)时间内解决 QTREE 问题的 LCA 算法。事实上本题存在更优秀的算法，本文就不继续论述了。

总结

RMQ 和 LCA 问题作为经典问题无论在算法学习上还是实际应用中都发挥了巨大的作用。山穷水复疑无路，柳暗花明又一村，在我们对题目毫无思路，思想阻塞之时，参考一下经典算法，利用已有的知识和模型，会大大加快解题的速度与精度。

站在巨人的肩膀上，才能看得更高，更远！

【参考文献】

【感谢】