**1、  概述**

LCA（Least Common Ancestors），即最近公共祖先，是指这样一个问题：在有根树中，找出某两个结点u和v最近的公共祖先（另一种说法，离树根最远的公共祖先）。 RMQ（Range Minimum/Maximum Query），即区间最值查询，是指这样一个问题：对于长度为n的数列A，回答若干询问RMQ（A,i,j）(i,j<=n)，返回数列A中下标在i，j之间的最小/大值。这两个问题是在实际应用中经常遇到的问题，本文介绍了当前解决这两种问题的比较高效的算法。

**2、  RMQ算法**

对于该问题，最容易想到的解决方案是遍历，复杂度是O(n)。但当数据量非常大且查询很频繁时，该算法也许会存在问题。

本节介绍了一种比较高效的在线算法（ST算法）解决这个问题。所谓在线算法，是指用户每输入一个查询便马上处理一个查询。该算法一般用较长的时间做预处理，待信息充足以后便可以用较少的时间回答每个查询。ST（Sparse Table）算法是一个非常有名的在线处理RMQ问题的算法，它可以在O(nlogn)时间内进行预处理，然后在O(1)时间内回答每个查询。

首先是预处理，用动态规划（DP）解决。设A[i]是要求区间最值的数列，F[i, j]表示从第i个数起连续2^j个数中的最大值。例如数列3 2 4 5 6 8 1 2 9 7，F[1，0]表示第1个数起，长度为2^0=1的最大值，其实就是3这个数。 F[1，2]=5，F[1，3]=8，F[2，0]=2，F[2，1]=4……从这里可以看出F[i,0]其实就等于A[i]。这样，DP的状态、初值都已经有了，剩下的就是状态转移方程。我们把F[i，j]平均分成两段（因为f[i，j]一定是偶数个数字），从i到i+2^(j-1)-1为一段，i+2^(j-1)到i+2^j-1为一段(长度都为2^（j-1）)。用上例说明，当i=1，j=3时就是3,2,4,5 和 6,8,1,2这两段。F[i，j]就是这两段的最大值中的最大值。于是我们得到了动态规划方程F[i, j]=max（F[i，j-1], F[i + 2^(j-1)，j-1]）。

然后是查询。取k=[log2(j-i+1)]，则有：RMQ(A, i, j)=min{F[i,k],F[j-2^k+1,k]}。 举例说明，要求区间[2，8]的最大值，就要把它分成[2,5]和[5,8]两个区间，因为这两个区间的最大值我们可以直接由f[2，2]和f[5，2]得到。

算法伪代码：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | //初始化    INIT\_RMQ    //max[i][j]中存的是重j开始的2^i个数据中的最大值，最小值类似，num中存有数组的值    for i : 1 to n      max[0][i] = num[i]    for i : 1 to log(n)/log(2)      for j : 1 to (n+1-2^i)         max[i][j] = MAX(max[i-1][j], max[i-1][j+2^(i-1)]    //查询    RMQ(i, j)    k = log(j-i+1) / log(2)    return MAX(max[k][i], max[k][j-2^k+1]) |

当然，该问题也可以用线段树（也叫区间树）解决，算法复杂度为：O(N)~O(logN)，具体可阅读这篇文章：《[数据结构之线段树》](http://dongxicheng.org/structure/segment-tree/)。

**3、  LCA算法**

对于该问题，最容易想到的算法是分别从节点u和v回溯到根节点，获取u和v到根节点的路径P1，P2，其中P1和P2可以看成两条单链表，这就转换成常见的一道面试题：【判断两个单链表是否相交，如果相交，给出相交的第一个点。】。该算法总的复杂度是O（n）（其中n是树节点个数）。

本节介绍了两种比较高效的算法解决这个问题，其中一个是在线算法（DFS+ST），另一个是离线算法（Tarjan算法）。

**在线算法DFS+ST描述**(思想是：将树看成一个无向图，u和v的公共祖先一定在u与v之间的最短路径上)：

（1）DFS：从树T的根开始，进行深度优先遍历（将树T看成一个无向图），并记录下每次到达的顶点。第一个的结点是root(T)，每经过一条边都记录它的端点。由于每条边恰好经过2次，因此一共记录了2n-1个结点，用E[1, ... , 2n-1]来表示。

（2）计算R：用R[i]表示E数组中第一个值为i的元素下标，即如果R[u] < R[v]时，DFS访问的顺序是E[R[u], R[u]+1, …, R[v]]。虽然其中包含u的后代，但深度最小的还是u与v的公共祖先。

（3）RMQ：当R[u] ≥ R[v]时，LCA[T, u, v] = RMQ(L, R[v], R[u])；否则LCA[T, u, v] = RMQ(L, R[u], R[v])，计算RMQ。

由于RMQ中使用的ST算法是在线算法，所以这个算法也是在线算法。

【举例说明】

T=<V,E>，其中V={A,B,C,D,E,F,G},E={AB,AC,BD,BE,EF,EG},且A为树根。则图T的DFS结果为：A->B->D->B->E->F->E->G->E->B->A->C->A，要求D和G的最近公共祖先, 则LCA[T, D, G] = RMQ(L, R[D], R[G])= RMQ(L, 3, 8)，L中第4到7个元素的深度分别为：1,2,3,3，则深度最小的是B。

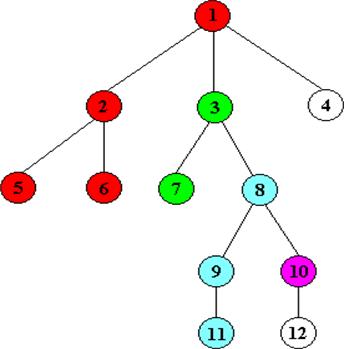
**离线算法（Tarjan算法）描述**：

所谓离线算法，是指首先读入所有的询问（求一次LCA叫做一次询问），然后重新组织查询处理顺序以便得到更高效的处理方法。Tarjan算法是一个常见的用于解决LCA问题的离线算法，它结合了深度优先遍历和并查集，整个算法为线性处理时间。

Tarjan算法是基于并查集的，利用并查集优越的时空复杂度，可以实现LCA问题的O(n+Q)算法，这里Q表示询问 的次数。更多关于并查集的资料，可阅读这篇文章：《[数据结构之并查集](http://dongxicheng.org/structure/union-find-set/)》。

同上一个算法一样，Tarjan算法也要用到深度优先搜索，算法大体流程如下：对于新搜索到的一个结点，首先创建由这个结点构成的集合，再对当前结点的每一个子树进行搜索，每搜索完一棵子树，则可确定子树内的LCA询问都已解决。其他的LCA询问的结果必然在这个子树之外，这时把子树所形成的集合与当前结点的集合合并，并将当前结点设为这个集合的祖先。之后继续搜索下一棵子树，直到当前结点的所有子树搜索完。这时把当前结点也设为已被检查过的，同时可以处理有关当前结点的LCA询问，如果有一个从当前结点到结点v的询问，且v已被检查过，则由于进行的是深度优先搜索，当前结点与v的最近公共祖先一定还没有被检查，而这个最近公共祖先的包涵v的子树一定已经搜索过了，那么这个最近公共祖先一定是v所在集合的祖先。  
算法伪代码：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | LCA(u)  {  Make-Set(u)  ancestor[Find-Set(u)]=u  对于u的每一个孩子v  {  LCA(v)  Union(u,v)  ancestor[Find-Set(u)]=u  }  checked[u]=true  对于每个(u,v)属于P // (u,v)是被询问的点对  {  if checked[v]=true    then {  回答u和v的最近公共祖先为ancestor[Find-Set(v)]  }        }  } |

【举例说明】

根据实现算法可以看出，只有当某一棵子树全部遍历处理完成后，才将该子树的根节点标记为黑色（初始化是白色），假设程序按上面的树形结构进行遍历，首先从节点1开始，然后递归处理根为2的子树，当子树2处理完毕后，节点2， 5， 6均为黑色；接着要回溯处理3子树，首先被染黑的是节点7（因为节点7作为叶子不用深搜，直接处理），接着节点7就会查看所有询问(7, x)的节点对，假如存在(7, 5)，因为节点5已经被染黑，所以就可以断定(7, 5)的最近公共祖先就是find(5).ancestor，即节点1（因为2子树处理完毕后，子树2和节点1进行了union，find(5)返回了合并后的树的根1，此时树根的ancestor的值就是1）。有人会问如果没有(7, 5)，而是有(5, 7)询问对怎么处理呢? 我们可以在程序初始化的时候做个技巧，将询问对(a, b)和(b, a)全部存储，这样就能保证完整性。

**4、  总结**

LCA和RMQ问题是两个非常基本的问题，很多复杂的问题都可以转化这两个问题解决，这两个问题在ACM编程竞赛中遇到的尤其多。这两个问题的解决方法中用到很多非常基本的数据结构和算法，包括并查集，深度优先遍历，动态规划等。

**5、  参考资料**

（1）      [判断两个链表是否相交](http://blog.csdn.net/ldong2007/archive/2009/09/11/4544203.aspx)

（2）       [博文《LCA问题（含RMQ的ST算法）》](http://www.cppblog.com/Icyflame/archive/2009/07/04/88987.html)

（3）      [博文《Range Minimum Query and Lowest Common Ancestor》](http://www.topcoder.com/tc?module=Static&d1=tutorials&d2=lowestCommonAncestor#Range_Minimum_Query_%28RMQ%29)

（4）       [博文《LCA问题（最近公共祖先问题）+ RMQ问题》](http://ayzk.wordpress.com.cn/archives/14)

（5）       [博文《最近公共祖先(LCA)的Tarjan算法》](http://hi.baidu.com/windog18/blog/item/10c6d2102f5a8df4c3ce79fa.html)

（6）       [博文《LCA 最近公共祖先的Tarjan算法》](http://my.chinaunix.net/space.php?uid=1721137&do=blog&id=181005)

———————————————————————————————-

更多关于数据结构和算法的介绍，请查看：[数据结构与算法汇总](http://dongxicheng.org/structure/structure-algorithm-summary/)

———————————————————————————————-