Rmq问题，所谓rmq问题，就是（range max/min query），就是区间最大值，最小值的查询。

最常规的方法想到的是直接的查询，就是从扫面一边数组中的相关位置，然后找到最值。时间复杂度为O(n)， 但是这样的算法，如果一旦查询的次数很多，查询的量很大的话，那么会变得效率不是很高。

这篇将rmq的博文，大致分两部分，第一部分讲一下一个在线的rmq算法，叫做ST(Sparse Table)算法，是采用动态规划的思想。第二部分呢，是说一下，LCA与RMQ问题的转化，LCA问题可以经过DFS+st变成rmq（min）的解法，而且求解的时间复杂度规模是一样的。那么下面先来开始第一部分：

St算法，st算法是采用动态规划的思想。我们以最小值为例，现在要求RMQ（A,i,j）,数组A就是我们要求区间最值的数组，我们用M[i][j]表示从i开始，长度为2^j的数组长度内最小值的数组元素的下标。如果我们要这个区间内的最小值，我们可以将其转化为寻找M[i][j-1]与M[i+2^(j-1)-1][j-1]的最小值，因为这两个区间刚好是上述区间的一半。这样我们就得到了状态转移方程，

M[i][j]=M[i][j-1] (A[M[i][j-1]]<=A[M[i+2^(j-1)][j-1]])

Else M[i][j] = M[i+2^(j-1)][j-1]

Dp不仅需要状态转移方程，并且还需要初始值，那么我们的初始值该如何得出呢。其实我们不难看出M[i][0] = i;

那么这样，我们就轻松得求出了一段代码了。

void process2(int M[MAXN][LOGMAXN], int A[MAXN], int N)

  {

      int i, j;

     //initialize M for the intervals with length 1

      for (i = 0; i < N; i++)

          M[i][0] = i;

     //compute values from smaller to bigger intervals

      for (j = 1; 1 << j <= N; j++)

          for (i = 0; i + (1 << j) - 1 < N; i++)

              if (A[M[i][j - 1]] < A[M[i + (1 << (j - 1))][j - 1]])

                  M[i][j] = M[i][j - 1];

              else

                  M[i][j] = M[i + (1 << (j - 1))][j - 1];

  }

接下来我们要进行查询了。那么我们的想法是找到一个是找到两块空间，使其可以完全得覆盖i到j。我们找了k=[log2(j-i+1)],中括号是取整的意思。

那么我们就可以得出结论：

RMQ(A,i,j) = M[i][k] (A[M[i][k]]<A[M[j-2^k+1][k]])

else RMQ(A,i,j) = M[A[j-2^k+1][k]];

第二部分，按照计划，讨论一下LCA与RMQ最小值问题的相互转化

算法的过程是：比方说，现在我我们有一棵树，我们使用dfs对这个树来进行访问，在这个树访问的过程中，我们需要记录几个量。访问时每访问到一个节点（无论是初次访问还是回溯时访问的），首先我们要记录下这个点所在的层数，也就是说是这个点在数中的深度，有这样的话，我们会访问到2n-1（加入树的节点个数为n的话）个节点，那么我们新建立一个数组叫做E,大小为2n-1.然后我们还要开一个数组R，大小也为2n-1.犹豫dfs访问我们会记录一个顺序，那么比方说我们的dfs顺序为A->B->A->c->A( 这个就是A为根，并且两个节点为B，C的简单的树)，那么R[i]记录的就是第i个点在序列中第一次出现的位置。然后用D[2n-1]记录对应访问顺序的节点的所在的层数，下面从别人的博客中截取一个小的例子做示范一下：

  (1)  
  
    / \

(2)   (7)

/ \     \

(3) (4)   (8)

    /   \

(5)    (6)

一个nlogn 预处理，O(1)查询的算法.

Step 1:

        按先序遍历整棵树，记下两个信息:结点访问顺序和结点深度.

        如上图:

        结点访问顺序是: 1 2 3 2 4 5 4 6 4 2 1 7 8 7 1 //共2n-1个值

        结点对应深度是: 0 1 2 1 2 3 2 3 2 1 0 1 2 1 0

Step 2:

        如果查询结点3与结点6的公共祖先,则考虑在访问顺序中

        3第一次出现，到6第一次出现的子序列: 3 2 4 5 4 6.

        这显然是由结点3到结点6的一条路径.

        在这条路径中，深度最小的就是最近公共祖先(LCA). 即

        结点2是3和6的LCA.

Step 3:

        于是问题转化为, 给定一个数组D,及两个数字i,j,如何找出

        数组D中从i位置到j位置的最小值..

        如上例,就是D[]={0,1,2,1,2,3,2,3,2,1,0,1,2,1,0}.

        i=2;j=7;

        接下来就是经典的RMQ问题.

弄明白了这些问题后，下面我们来研究一下RMQ的算法的复杂度的问题，可以看到在第一部分代码的双重循环中，最外一层循环是移位操作。所以其复杂度应该是logN，而里面一层的复杂度是N，所以总的算法的复杂度是N\*logN。并且这个复杂度只是进行初始化时的操作，在初始化结束之后，每次询问只要O(1)的复杂度。

在来看一下，LCA转化后的算法，一次dfs，应该是logN的复杂度，然后生成了一个2\*N-1的数组，然后运用RMQ st的算法，复杂度变为（2N-1）\*log（2N-1），    其实最终还是N\*logN的。而相比之下，上一篇文章中所提到的Tarjan算法的复杂度为O(n+Q)，其中Q为查询的次数。