RMQ 和 LCA

【内容简介】

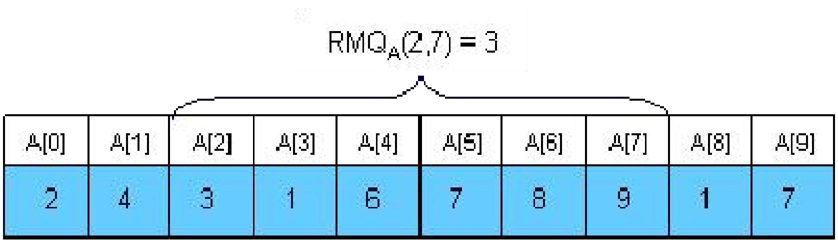
讨论解决 RMQ、LCA 问题的算法、以及 RMQ、LCA 两类问题的相互转换。它们在字符串处理和生物学计算中有着广泛应用，在信息学奥赛中更是被广泛应用和扩展，所以熟练掌握 RMQ 和 LCA 问题就显得十分重要。

※ 在内容开始，我们假设一个算法预处理时间为 **f(n)**，查询时间为 **g(n)**。这个算法复杂度的标记为 **<f(n)，g(n)>**(或 **f(n)-g(n)**)。

【关键词】RMQ、LCA、ST 算法、Tarjian 离线算法、±1RMQ、笛卡尔树、欧拉序列、并查集

一、RMQ(Range Minimum/Maximum Query)问题：

RMQ 问题是求给定区间中的最值问题，如下图所示：



RMQ 问题（图中记录的是最小值的位置）

当然，最简单的算法是 O(n)的，但是对于查询次数很多 m（假设有 100 万次），则这个算法的时间复杂度为 O(mn)，显然时间效率太低。可以用线段树将查询算法优化到 O（logn)（在线段树中保存线段的最值），而线段树的预处理时间复杂度为 O(n)，线段树整体复杂度为<O(n), O(logn)>。不过，Sparse_Table 算法（简称 ST 算法）才是最好的：它可以在 O(nlogn)的预处理以后实现 O(1)的查询效率，即整体时间复杂度为<O(nlogn),O(1)>。

1.1 ST 算法（★★★★）

ST 算法，即 Sparse Table 算法。下面把 ST 算法分成预处理和查询两部分来说明(以求最小值为例)，它的时间复杂度为<O (nlogn), O(1)>。

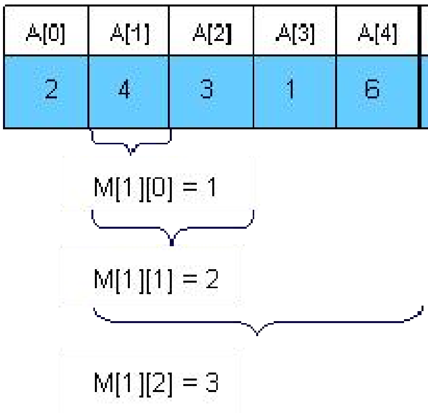
1.1.1 预处理:

预处理使用 DP 的思想，f(i, j)表示[i, i+2^j - 1]区间中的最小值，即 f[i,j]表示从第 i 个数起连

续 2^j 个数中的最小值。我们可以开辟一个数组专门来保存 f(i, j)的值。

例如，f(1, 0)表示[1,1]之间的最小值,就是 num[1]；f(1, 2)表示[1, 4]之间的最小值, f(2, 4)表示[2, 17]之间的最小值。

注意, 因为 f(i, j)可以由 f(i, j - 1)和 f(i+2^(j-1), j-1)导出, 而递推的初值(所有的 f(i, 0)=num[i])都是已知的。所以我们可以采用自底向上的算法递推地给出所有符合条件的 f(i, j)的值。



ST 算法(图中记录的是最小值的位置)

ST 算法的状态转移方程：

$$f(i,j)=\begin{cases} a[i] \quad , & j=0 \\ \min(f(i,j-1), f(i+2^{j-1}, j-1), & j>0 \end{cases}$$

例如：f(2,3) 保存的是 a[2],a[3],a[4],……,a[9] 中的最小值，而 f(2,3)=min(f(2,2),f(6,2))=min((a[2],a[3],a[4],a[5]),(a[6],a[7],a[8],a[9]))

1.1.2 查询:

假设要查询从 m 到 n 这一段的最小值, 那么我们先求出一个最大的 k, 使得 k 满足 $2^k \le (n-m+1)$，于是我们就可以把[m, n]分成两个(部分重叠的)长度为 2^k 的区间: $[m, m+2^k-1]$, $[n-2^k+1, n]$;

而我们之前已经求出了 $f(m,k)$ 为 $[m, m+2^k-1]$ 的最小值, $f(n-2^k+1,k)$ 为 $[n-2^k+1, n]$ 的最小值。

我们只要返回其中更小的那个, 就是我们想要的答案, 这个算法的时间复杂度是 O(1)的。

```
k= trunc(ln(r-l+1)/ln(2)); // 求[l,r]之间的最小值

ans:=max(F[l, k],F[r-2^k+1,k]);
```

例如, rmq(1,12) = min(f(1,3), f(5,3)) ($k=\left|\log_2(12-1+1)\right|=3$)

ST 算法的 O(1)查询（有部分重叠）

RMQ 区间最大值:【参考代码】(部分)

```
function max(x,y: longint): longint;
begin
 max:=x;
 if y>x then max:=y;
end;
```

```
function query(s,t: longint): longint;  // 查询[s,t]间的最大值
var
 k: longint;
begin
 k:=trunc(ln(t-s+1)/ln(2));
 query:=max(a[s,k],a[t-(1<<k)+1,t]);   // 即max(a[s,k],a[t-2^k+1,t])
end;
```

```
procedure init;
var
 i,j,p: longint;
begin
 assign(input,'rmq-st.in');
 reset(input);
 readln(n);
 for i:=1 to n do read(a[i,0]);
 p:= trunc(ln(n)/ln(2));
 for j:=1 to p do       // a[i,j]表示从i开始,2^j个元素的最大值
  for i:=1 to (n-(1<<j)+1) do    // n-(1<<j)+1 即 n-2^j+1
   a[i,j]:=max(a[i,j-1],a[i+1<<(j-1),j-1]);
 close(input);
end;
```

1.2 巧妙易实现的分块 -- sqrt(n)算法：（★★★）

把数组分割成 sqrt(n)大小的段。用一个数组 M[1..sqrt(n)]为每一段保存最小值(或位置)。M 可以很容易在 O(n)时间内完成预处理。例如：

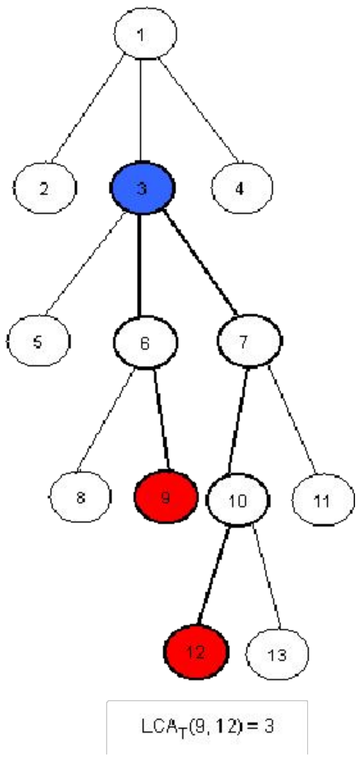
		RMQ(3,8)=1							
A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]
2	4	3	1	6	7	8	9	1	7
M[1]=2			M[2]=1			M[3]=1			

如在上例中，计算 RMQ(3,8)的值，我们应该比较 A[3]、M[2]、A[7]、A[8]值即可。可以很容易看出这个算法每一次查询不会超过 3*sqrt(n)次操作。

这个方法非常容易实现，并且还可以将它改成问题的动态版本（边查询、边改元素）。

二、LCA(Lowest Common Ancestor)问题：

LCA（最近公共祖先）问题，即给定一棵树 T 和两个节点 u 和 v，找出 u 和 v 的离根节点最远的公共祖先，如右图。



2.1 在线算法（★★★）

类似 RMQ 的 ST 算法，每次询问 O(logN)，d[i]表示 i 点的深度，p[i,j] 表示 i 的 2^j 倍祖先，那么就有一个递推式子 p[i,j]=p[p[i,j-1],j-1]。这样一个 O(NlogN)的预处理求出每个节点的 2^k 的祖先，时间复杂度<O(nlogn, O(logn)>。

$$p[i,j]=\begin{cases} T[i] & j=0 \\ p[\,p[i,j-1],j-1] & j>0 \end{cases}$$

【注】p[i,0]指向 i 的父结点(T[i])。

然后对于每一个询问的点对 a, b 的最近公共祖先就是：

① 先判断是否 d[a] > d[b] ，如果是的话就交换一下(保证 a 的深度小于 b，方便下面的操作)。

② 然后把 b 调到与 a 同深度，同深度以后再把 a, b 同时往上调 dec(k)，调到有一个最小的 k 满足 p[a,k]<>p[b,k] (a=p[a,k],b=p[b,k]它们是在不断更新的)。

③ 最后 p[a,0]或 p[b,0]就是他们的最近公共祖先。

以右图为例 LCA(9，12)我们来看看这个算法的实现过程：

① 求得 $k=\lfloor\log_2 d(12)\rfloor=\lfloor\log_2 5\rfloor=2$，

求得 p[12,0]=10， p[12,1]=7， p[12,2]=1

P[9,0]=6， p[9,1]=3，

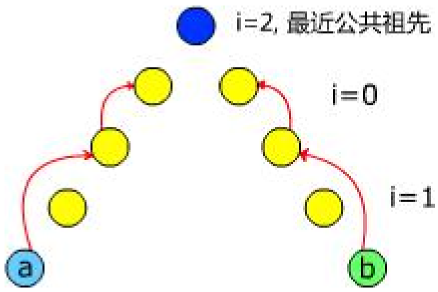
② d(9)<d(12)，找到与 9 同层的结点 10（12 的祖先），LCA（9,12）转化为 LCA(9,10)

③ 类似二分搜索的方式向上爬树，最终到达结点 6,结点 7。

```
a:=9; b:=10;
for i:=k downto 0 do
 if p[a,i]<>p[b,i] then
     begin
        a:= p[a,i];
        b:= p[b,i];
     end;
```

④ 最终 p[6,0]（或 p[7,0]）的值为 3,即是结点 9,12 的最近公共祖先。

【注】如下图求 LCA(a,b)，当 i=2 时，p[a,2]=p[b,2]（蓝色小圆)；i 递减变为 1，此时 p[a,1]<>p[b,1]，将 a:=p[a,1],b:=p[b,1]，完成一次爬树；继续 i 递减变为 0，此时 p[a,0]<>p[b,0]，再将 a:=p[a,0],b:=p[b,0]，再向上完成一次爬树，循环结束；最终 p[a,0]（或 p[b,0]）即是它们的最近公共祖先。



求最近公共祖先

【算法实现】

首先我们构建一张表p[1..N,1..logN]，这里P[i][j]指的是结点i的第2^j个祖先。

```
procedure dp;
var
 i,j,st: longint;
begin
 // 初始化
 st:= trunc(ln(n)/ln(2));
 for i:=1 to n do
  for j:=0 to st do
    p[i,j]:=-1;
```

```
 for i:=1 to n do p[i,0]:=t[i]; // 指向父结点

 // dp求p[i,j],p[i,j]表示i的第2^j个祖先,j=0表示父结点
 for j:=1 to st do
  for i:=1 to n do
    if p[i,j]<>-1 then p[i,j]:=p[p[i,j-1],j-1];
end;
```

```
function lca(a,b: longint);
var
 tmp,log,i: longint;
begin
 if (L[a] < L[b]) then
  begin
    tmp:= a;
    a:=b;
    b:=tmp;
  end;

  //计算log(L[a]
  log:= trunc(ln(L(a)-1)/ln(2)));

  //找到与b同层的结点
  for i:= log downto 0 do
    if (L[a] - (1 << i) >= L[b])  // 即L(a)-2^i>=L(b)
             a:= p[a,i];    // L(a)+2^i(上移到这层)
    if (a = b) then exit(a); // a是b的子树

  // 计算LCA(a,b)
  for i:= log downto 0 do
    if (p[a,i] <> -1) and (p[a,i]<>p[b,i]) then
     begin
       a:= p[a,i];
       b:= p[b,i];
     end;
  exit(t[a]);
end;
```

2.2 离线算法－Tarjian（★★★★）

Tarjan 算法是由 Robert Tarjan 在 1979 年发现的一种高效的离线算法，也就是说，它要首先读入所有的询问（求一次 LCA 叫做一次询问），然后并不一定按照原来的顺序处理这些询问，而打乱这个

顺序正是这个算法的巧妙之处，该算法的时间复杂度 $O(n*\alpha(n)+Q)$，其中 $\alpha(x)$，对于 x=宇宙中原子数之和，$\alpha(x)$不大于 4，N 表示问题规模，Q 表示询问次数。

首先需要有一些预备知识：

1.基本图论

2.并查集

（详见维基百科：http://zh.wikipedia.org/wiki/%E5%B9%B6%E6%9F%A5%E9%9B%86）

这里提一下并查集的概念，并查集是一种处理元素之间等价关系的数据结构，一开始我们假设元素都是分别属于一个独立的集合里的，主要支持两种操作：

(1) 合并两个不相交集合(Union)

(2) 判断两个元素是否属于同一集合(Find)

需要知道一点，就是并查集的 Find 操作的时间复杂度是常数级别的。

利用并查集优越的时空复杂度，我们可以实现 LCA 问题的 O(n+Q)算法，这里 Q 表示询问的次数。

Tarjan 算法基于深度优先搜索的框架，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。

【Tarjian 算法的伪代码】

```
function TarjanOLCA(u)
  MakeSet(u);
  u.ancestor := u;    // 将集合u的祖先指向自己
  for each v in u.children do  // DFS所有孩子
     TarjanOLCA(v);
     Union(u,v);     // 与根结点U合并（并查集操作）
     Find(u).ancestor := u;   // 将u所在集合根的祖先指向u
  u.colour := black;  // 当所有孩子都已遍历，则标记根已完成
  for each v such that {u,v} in P do  // 找所有与u相关的查询
    if v.colour == black  // 如果另一结点v是前面标记过的,则输出递归向上返回根的祖先
      print "Tarjan's Least Common Ancestor of " + u +
            " and " + v + " is " + Find(v).ancestor + ".";
```

```
function MakeSet(x)      // 创建集合x
     x.parent := x      // x的父结点指向自己
     x.rank   := 0      // 集合的rank值（类似于树的层数）

function Union(x, y)
     xRoot := Find(x)   // 集合x的根
     yRoot := Find(y)   // 集合y的根
     if xRoot.rank > yRoot.rank  // 集合rank值小的挂到rank值大的集合下
         yRoot.parent := xRoot
     else if xRoot.rank < yRoot.rank
         xRoot.parent := yRoot
     else if xRoot != yRoot // 避免出现已在同一集合中的元素进行合并
         yRoot.parent := xRoot
         xRoot.rank := xRoot.rank + 1

function Find(x)
// 并查集操作find,递归查找集合的根结点，并将沿途所有结点均指向根（路径压缩）
     if x.parent == x
        return x
     else
        x.parent := Find(x.parent)
        return x.parent
```

【解释】

① Union (x,y); // 并查集的合并操作

将集合 x 与集合 y 合并，使用 rank 值优化，将 rank 值较低的集合指向 rank 值较高的集合，以降低合并后的高度。

图 1　叶节点 v 与父节点 u

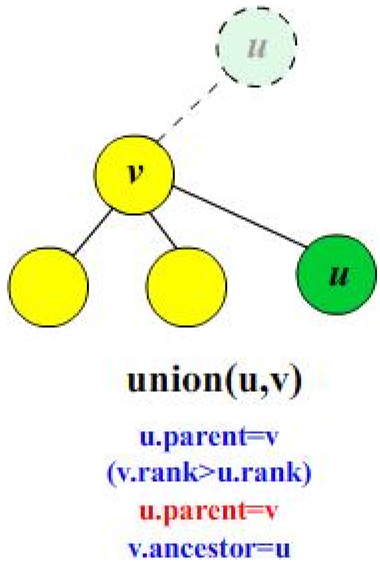


图 2　子树 v 与父节点 u

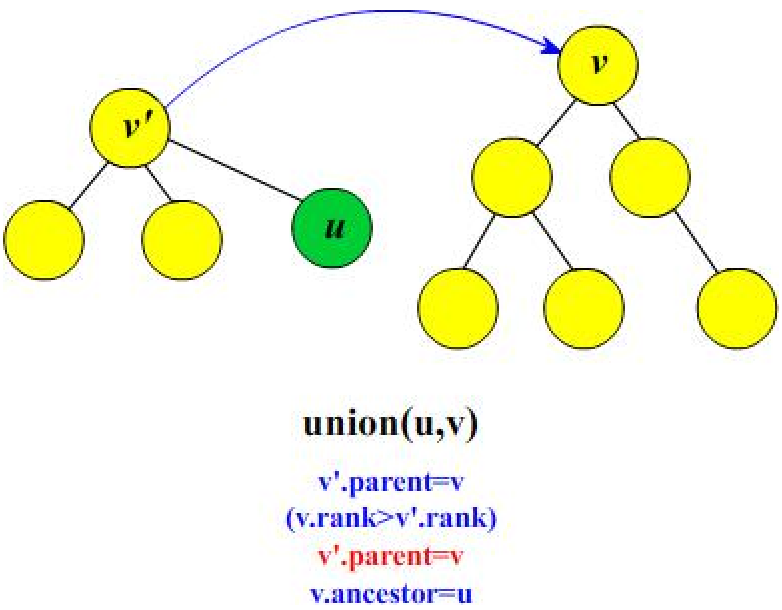


图 3　子树 v'与子树 v 合并(v 和 v'均为 u 的两棵子树)

每个节点的父节点(node.parent)和祖先节点(node.ancestor)是不同的：由于使用 rank 启发式函数，使集合 A 和集合 B 在合并时，先考虑集合 A、B 的各自 rank 值（层数），使较小 rank 值的集合挂在较大 rank 值的集合下面。

当一个节点 u 与它的子树 v 合并时，u.rank<v.rank 时，u.parent=v，这样就改变了原有的父子关系，但必须要使 v.ancestor=u，也就是说子树最后指向的父节点可以不是它的根节点，但这棵子树的祖先节点一定是它的根节点，最后求的也是它的祖先节点。使用 rank 的目的，也是尽量使得合并后树的高度较低，提高 find(v)的查找效率。

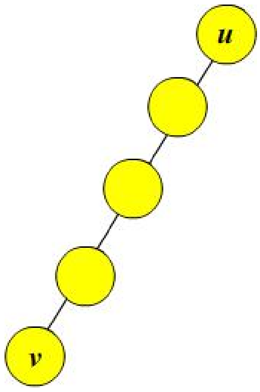
② `find(u).ancestor:=u;`

查找 u 的当前根结点，然后再将此结点祖先设为 u。这样就使得合并后集合中的所有结点 x，经过 find(x)，均能找到共同的根结点 v，通过 v.ancestor 值求得它的祖先值。如图 2、图 3 中祖先 ancestor 均是 u（根），但合并后的根节点都是 v 了。

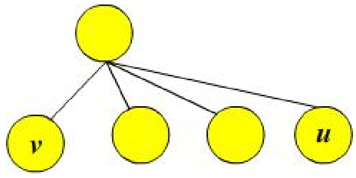
③ rank ：优化合并集合的操作

原则是将两个集合（树）合并时，将 rank 值小的（即层数较少的树）挂到 rank 值大的树下。这样就可以使得合并后的集合（树）的各结点的 rank 值和较小，这样在使用 find(x)时，能够快速到达树根，求得它的祖先值。

如果不使用 rank 值优化，对于下图的数据就会出现单链的情况，这样使得 find(x)效率大大降低。



单链



使用 rank 值优化

三、RMQ 转换为 LCA 问题

RMQ 转换为 LCA，需要借助“笛卡尔树”。

- ## 笛卡尔树(Cartesian Tree):

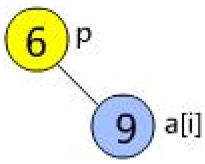
“笛卡尔树”是一种特殊的堆，它根据一个长度为 n 的数组 A 建立。它的根是 A 的最小元素位置 i,而左子树和右子树分别为 A[1…i-1]和 A[i+1…n]的笛卡尔树。中序遍历笛卡尔树就可以得到原数列。

笛卡儿树的重要应用之一是实现 RMQ 问题向 LCA 问题的转化。

Cartesian 树的建立算法：

首先，先从 A[1]开始建立，以后每次加一个数 A[i]，就修改 Cartesian 树。不难发现，这个数一定在这棵树的最右边的路径上。而且一定没有右孩子，所以，只沿着最右路径自底向上把各个节点 p 和 A[i]做比较。

① A[i]>p ，那么 A[i]就为 p 的右孩子；

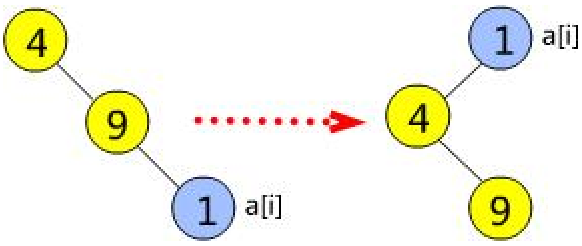


A[i]>p

② A[i]<p ，但 A[i]>p 的父亲，那么 A[i]为 p 的父亲的右孩子，而 p 则改为 A[i]的左孩子；否则继续上升。

A[i]<p, A[i]>p.parent

③ A[i]<树根，则以 A[i]为根，原树根为 A[i]的左孩子。



A[i]>tree.root

因为每个节点最多进入和退出最右路径各一次，所以，均摊时间复杂度为 O(n)。

【定理】数组 A 的 Cartesian 树记为 C(A)，则

RMQ(A, i, j) = LCA(C(A), i, j)

例如：有数组 A=(7、5、8、1、10)，则它建立的笛尔树如图 3.1 所示。

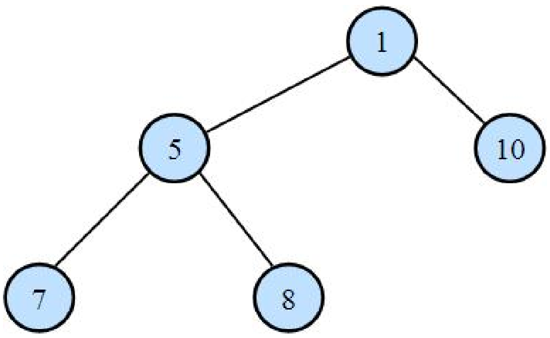
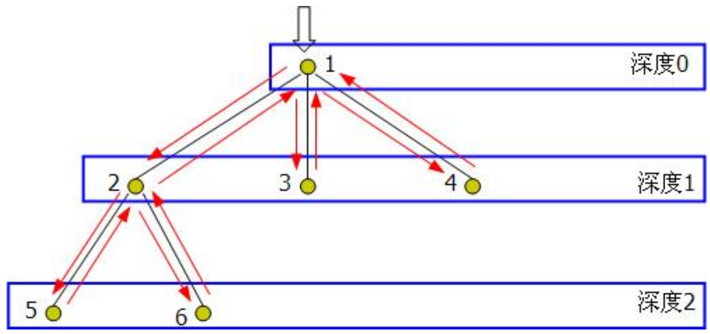


图 3.1 建立的笛卡尔树

四、LCA 转换为 RMQ 问题

1. 对有根树 T 进行 DFS，将遍历到的结点按照顺序记下，我们将得到一个长度为 2N – 1 的序列，称之为 T 的欧拉序列 F。



DFS 遍历

欧拉序列 F	1	2	5	2	6	2	1	3	1	4	1
深度序列 B	0	1	2	1	2	1	0	1	0	1	0

2．每个结点都在欧拉序列中出现，我们记录结点 u 在欧拉序列中第一次出现的位置为 pos(u)。

u	1	2	3	4	5	6
pos(u)	1	2	8	10	3	5

3．根据 DFS 的性质，对于两结点 u、v，从 pos(u)遍历到 pos(v)的过程中经过 LCA(u, v)有且仅有一次，且深度是深度序列 B[pos(u)…pos(v)]中最小的，并且问题规模仍然是 O(N)的。

LCA(T, u, v) = RMQ(B, pos(u), pos(v))

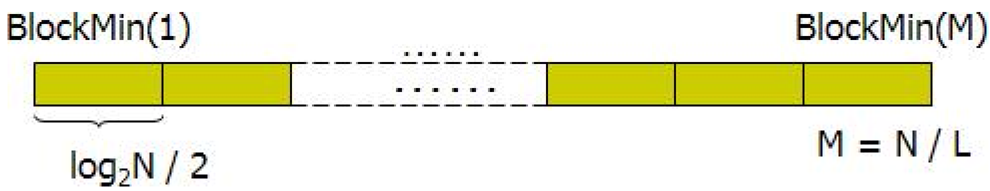
五、±1RMQ 算法

将 LCA 问题转换为 RMQ 问题后，利用 ST 算法来求解，时间复杂度是 O(nlogn)-O(1)，反而不及直接利用 Tarjian 算法求解 LCA 问题时间效率高。下面将介绍±1RMQ 算法，它求解的时间复杂度为 O(n)-O(1)，而上述公式中 **RMQ(B, pos(u), pos(v))** 深度序列 B 数组中任意相邻两个数的差是±1，正好利用“±1RMQ”算法来求解。

- ±1RMQ 算法

1．核心思想（分块）：

以 $L = \frac{\log_2 n}{2}$ 块长把 B 划分为 M = N / L 段，记录第 k 块的最小元素为 BlockMin(k)，把 M 块的最小值组成序列 Blocks，利用分块思想，我们可以把询问分为两个部分询问：

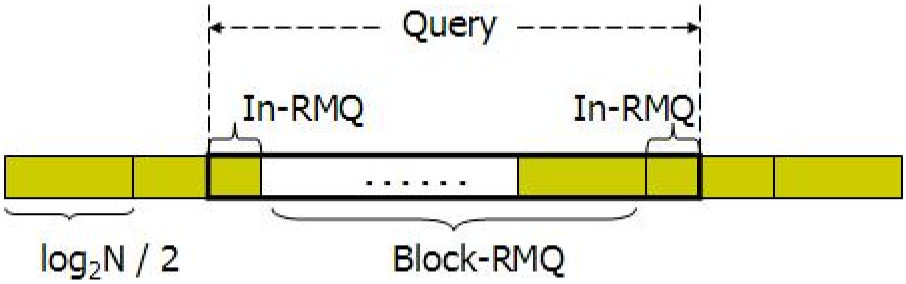


2. 求解方法：

⑴ 连续的 BlockMin 取最小值，即 Block-RMQ；（使用 ST 算法）

⑵ 两端块中某一部分取最小值，即 In-RMQ；（预处理所有可能依次回答）

这两个问题都可以 O(N) – O(1)内实现。



【有关±1RMQ 算法详细说明】

① 由于在±1RMQ 数组中相邻元素差值为 1，即|A[i]-A[i+1]|=1，如果设 a[0]=a[1]-1（或 a[1]+1），b[1]=a[1]-a[0]，b[2]=a[2]-a[1]，b[3]=a[3]-a[2]，…，b[i]=a[i]-a[i-1]。

可以得知：数组 b 中的元素只有两种，1 和-1。

这样

∵ b[1]+b[2]+b[3]+…+b[i]

= a[1]-a[0]+a[2]-a[1]+a[3]-a[2]+…+a[i]-a[i-1]

= a[i]-a[0]

∴ a[i]=b[1]+b[2]+…+b[i]+a[0]，即 b 数组前 i 项和，再加上 a[0]。

如 a 数组(5,6,5)，则定义 a[0]=a[1]-1=4(或 a[1]+1=6)，b[1]=5-4=1，b[2]=6-5=1，b[3]=5-6=-1。实际上求数组 a 某个区间的最小值，直接对 b 操作即可。

② Block-RMQ ：将 A 数组分成 l=[logn/2]的大小块，让 A'[i]为 A 中第 i 块的最小值，pos[i]为 A 中最小块值的位置。A'和 pos 数组的长度均为 n/l。我们利用 ST 算法预处理 A'数组，根据 ST 算法的预处理时间复杂度 O(nlogn)，n 为问题规模。这样将花费 O(n/l*log(N/l))=O(n)的时间和空间。经过预处理之后，我们可以在 O(1)时间内在很多块上进行查询。

③ In-RMQ：由于每个块的长度为 l=[logn/2]（值非常小），经过①处理得到的数组 b（均为 1 和-1），那么对于每一块(长度为 l)的元素排列的可能性共有 $2^l = sqrt(n)$ 种，再对每一种排列情况预处理任意两个位置之间的最小值，将其保存在表 P 中，这个可以在 $O(sqrt(n)*l^2)$ 的时间和空间内解决。

预处理 A 中的每一块元素的排列情况，索引表 P，并且将其存储在数组 T[1..n/l]中，即 T[1]存储第一块元素排列情况所对应表 p 的值，其中 n/l 表示数组 A 共有的块数。这里有个技巧，由于 b

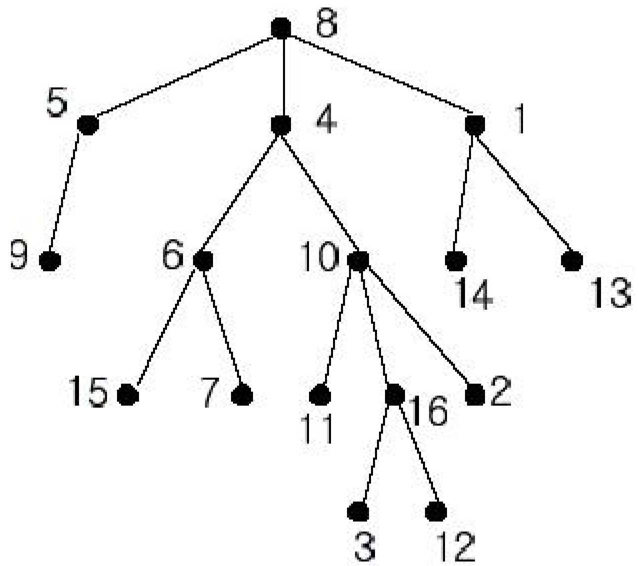
数组中的元素均为+1、-1，若将+1 替换为 1，-1 替换为 0，这样就转换为一个二进制数了。

五、上机编程题

1．北大 OJ 试题

① poj 1330：http://acm.pku.edu.cn/JudgeOnline/problem?id=1330

题目：Nearest Common Ancestors （ 求最近公共祖先 ）



Input

第 1 行 T，表示测试项目的组数；接下来的每一组测试数据的第 1 行是 N（2<=N<10,000），下面的 N-1 行，是树的 N-1 条边（有 N 个结点的树，必有 N-1 条边），每一组数据的最后一行是两个整数，表示求这两个结点的最近公共祖先。

Output

输出每一组问题的答案，每个答案占一行。

Sample Input

```
2
16
1 14
8 5
10 16
5 9
4 6
8 4
4 10
1 13
6 15
```

10 11
6 7
10 2
16 3
8 1
16 12
16 7
5
2 3
3 4
3 1
1 5
3 5

Sample Output

4
3

② poj 3264: http://acm.pku.edu.cn/JudgeOnline/problem?id=3264

题目：Balanced Lineup （ 平衡阵型 ）

在农场里有 N(1 ≤ *N* ≤ 50,000)头牛排成一队，每头牛的身高 height(1 ≤ *height* ≤ 1,000,000)，有 Q 次询问(1 ≤ *Q* ≤ 200,000)，求解区间内最高牛和最矮牛之间的身高差。

Time Limit: 5000MS **Memory Limit:** 65536K

Input

Line 1: N and Q.
Lines 2..N+1: Line i+1 表示第 i 头牛的身高
Lines N+2..N+Q+1: 两个整数 A 和 B (1 ≤ A ≤ B ≤ N)，代表求解的区间。

Output

输出每个区间内最高牛和最矮牛之间的身高差

Sample Input

6 3
1
7
3
4
2
5
1 5
4 6
2 2

Sample Output

6

3
0

③ poj 3368: http://acm.pku.edu.cn/JudgeOnline/problem?id=3368

题目：Frequent values

有 n 个整数，排列成 a1,a2,a3,…,an 非递减顺序。除此之外，还有一些询问 i 和 j (1 ≤ i ≤ j ≤ n)，对于每次查询，输出 ai 至 aj 区间内重复出现最多的次数（频度）。

Time Limit: 2000MS **Memory Limit:** 65536K

Input

输入数据包含多组测试，每组测试的开始包括两个整数 **n** 和 **q** ($1 \le n, q \le 100000$). 在下一行包括 **n** 整数 $a_1, \dots, a_n$ ($-100000 \le a_i \le 100000$, 每个 $i \in \{1, \dots, n\}$) ，每个数之间用安空格分隔。每个 i $\in \{1, \dots, n-1\}$: $a_i \le a_{i+1}$. 接下来的 **q** 行包含每次查询，由两个整数 **i** 和 **j** ($1 \le i \le j \le n$)组成。

最后一组测试的下一行，只有一个整数 0（结束）。

Output

输出每次询问的最大频度值

Sample Input

```
10 3
-1 -1 1 1 1 1 3 10 10 10
2 3
1 10
5 10
0
```

Sample Output

```
1
4
3
```

④ POJ 1986: http://acm.pku.edu.cn/JudgeOnline/problem?id=1986

题目：Distance Queries

有 m 个农场，有 q 次询问两个农场之间的距离。

Time Limit: 2000MS **Memory Limit:** 65536K

Input

* Lines 1: m q

* Lines 2..1+M: 每行有 4 个值，F1,F2,L 和 D 描述一条路。F1,F2 是连接的两个农场，L 是距离，D 是一个字符，是 'N', 'E', 'S', 或者 'W' ，给出从农场 F1 到 F2 路的方向。

* Line 2+M: 一个整数 K. 1 <= K <= 10,000 （表示 K 次询问）
* Lines 3+M..2+M+K: 每行两个整数 a,b，表示求解农场 a 和农场 b 之间的距离。

Output

* Lines 1..K: 回答 K 次询问两个农场间的距离，每个数值占一行。

Sample Input

```
7 6
1 6 13 E
6 3 9 E
3 5 7 S
4 1 3 N
2 4 20 W
4 7 2 S
3
1 6
1 4
2 6
```

Sample Output

```
13
3
36
```

Hint

农场 2 和农场 6 距离 20+3+13=36 .

4．vijos 并查集专题试题：http://www.vijos.cn/

六、小结

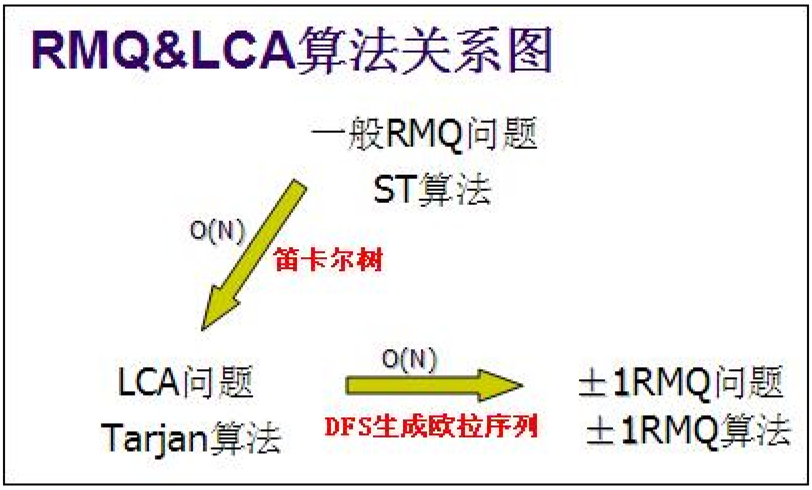
1．RMQ 和 LCA 问题算法比较：

算法名称	针对问题	时间消耗	空间消耗
ST算法	一般RMQ问题	$O(N\log_2 N)$-$O(1)$	$O(N\log_2 N)$
Tarjan算法	LCA问题	$O(N\alpha(N) + Q)$	O(N)
±1RMQ算法	±1RMQ问题	O(N)-O(1)	O(N)

注：N表示问题规模，Q表示询问次数

RMQ 和 LCA 问题算法比较

2．RMQ 和 LCA 算法关系如下图所示。



RMQ&LCA 算法关系图

3．本文参考及推荐资料：

① 刘汝佳《算法艺术与信息学奥赛》P55-57、P94

②《Range Minimum Query and Lowest Common Ancestor》by By danielp Topcoder Member 翻译：农夫三拳

③ 郭华阳《RMQ 与 LCA 问题》（国家集训队 2007 论文）