SIAM J. COMPUT.
Vol. 17, No. 6, December 1988

1988 Industria l 和应用数学学会
012

寻找最低共同祖先：简化和并行化*

BARUCIJ SCH IEBER't AN D UZI VISHKI N]4

摘要我们考虑以下问题。假设有一棵有根树 T 可供预处理。我们提出了一种线性时间和空间预处理算法，使我们能够在 O(1) 时间内回答每个查询，如 Harel 和 Tarjan ***[SIAM** J. Comput.,* 13 (1984), pp. ］我们的算法具有简单、易于并行化的优点。由此产生的并行预处理算法在EREW PRAM上使用最佳数量的处理器以对数时间运行。然后使用单个处理器在 O(l) 时间内回答每个查询。

关键词：并行算法、树算法、最小公有祖先

AMS（MOS）主题分类。05C05, 68Q10, 68Q20, 68R10

1. 导言我们考虑以下问题。给定一棵需要预处理的有根树 T(V, fi)，针对 T 中的任意一对顶点 x、*y*，回答形式为 "哪个顶点是 x 和 *y* 的最低共同祖先（LCA ）"的在线 LCA 查询（我们把这样的查询称为 LCA *(x,*y)）。我们提出了一种在串行 RAM 模型上以线性时间和线性空间运行的预处理算法。(随机存取机（RAM）模型的定义参见文献 [1]）。有了这种预处理算法，我们就能展示如何在恒定时间内处理每个这样的 LCA 查询。

我们还考虑了算法的并行化问题。我们使用的并行计算模型是独占-读取-独占-写入（EREW）并行随机存取机（PRAM）。PRAM 采用 *p* 个同步处理器，所有处理器都可以访问一个公共内存。EREW PRAM 不允许多个处理器同时访问同一内存位置，无论是读取还是写入。有关 PRAM 的研究结果，请参阅 [11]。

假设 Seq (n) 是已知顺序算法的最快最坏运行时间，其中 n 是当前问题的输入长度。使用 *p* 个处理器在 O(Seq ***(n)/p)*** 时间内运行的并行算法被称为***最优加速算法***，或者更简单地说，***最优算法。***并行计算的一个主要目标就是设计出运行速度尽可能快的最优算法。

我们的预处理算法很容易并行化，从而获得最优的并行预处理算法，在 EREW PRAM 上使用 n/log n 个处理器，运行时间为 O(log n)，其中 *n* 是 T 中的顶点数。

* 编辑于 1987 年 3 月 3 日收到；1988 年 2 月 22 日接受出版（修订版）。本研究得到了美国能源部能源研究办公室应用数学科学子项目的支持，合同编号为 DE-AC02-76ER03077。

i 特拉维夫大学数学科学学院计算机科学系，以色列特拉维夫，69975。现地址：T. J. Watson Research Center, IBM Research Division, Yorktown Heights, New York 10598。

I 纽约大学库兰特数学科学研究所计算机科学系，纽约州纽约市，邮编 10012。本文作者的研究得到了美国国家科学基金会（NSF-CCR-8615337）、美国海军研究办公室（N000I4-85-K-0046）以及以色列科学与人文研究院（Israel Academy of Sciences and Humanities）管理的电子、计算机和通信研究基金会（Foundation for Research in Electronics, Computers and Communication）的资助。

哈雷尔和塔尔扬在其详尽的论文[5]中针对同一问题给出了一种串行算法。他们的算法性能与我们的算法相同。不过，我们的算法有两个优点：(1) 在预处理阶段和查询处理阶段都要简单得多；(2) 它可以产生一种简单的并行算法。下面，我们将讨论与 [5] 算法的异同。*相似之处：* 这两种算法都使用了两个基本观点：(1) 可以在恒定时间内回答简单路径中的 LCA 查询；(2) 可以在恒定时间内回答完整二叉树中的 LCA 查询。这两种算法都将多个顶点的信息打包成一个 0(log *n)* 位的数字。*不同之处*这两种算法更微妙的部分在于展示如何利用上述两个观察结果来回答 LCA 查询。在这一部分，我们的方法完全不同。在预处理阶段，我们计算一个从输入树 T 的顶点到完整二进制 *B* 的顶点的映射。该映射有两个特性：(i) T 的所有顶点映射到 *B* 中的相同顶点形成一条路径；(ii) 对于 *T* 中的每个顶点 u，u 的后代映射到 u 在 *B 中的*映像的后代。另一方面，在 [5] 中，输入树 *T* 的顶点被映射到一棵高度为对数的任意树（称为压缩树）的顶点上。预处理包括对这棵压缩树进行相当复杂的处理。这种操作包括将压缩树划分为三层，分别对每层进行预处理，以及将压缩树嵌入一棵完整的二叉树中。

考虑一个动态 LCA 问题，在 LCA 查询中穿插着边的在线删除和插入。参考文献 [5] 也给出了该问题某些情况下的算法。本文不考虑这个问题。

我们的并行算法改进了以下结果。Tsin [9] 针对 LCA 问题给出了两种并行算法。在他的第一种算法中，预处理阶段和查询处理都需要对数时间，处理器数量为线性。在他的第二种算法中，使用 n 个^2 处理器，预处理阶段耗时为 0(log *n)*，使用单个处理器处理查询耗时为 0(1)。Vishkin [12] 提出了一种 LCA 问题的并行算法。处理一个 LCA 查询需要对数时间（如 **Tsin** 的第一种算法）。预处理阶段使用 n/log n 个处理器需要 O(log n)时间（与本文相同）。

请注意，**使用**我们的并行预处理算法，在允许读冲突的情况下，我们可以使用 (n + k)/log n 个处理器在 0(log n) 时间内处理 *k* 个l 行 LCA 查询。这影响了三个问题的并行算法性能：

(1) 给定一个无向图，定向其边，使得到的数图是强连接的（如果可以定向的话）[12]。(2) 计算双连接图的开耳分解和 st 编号 [8]。利用[3]中新的并行连接和列表排序算法，只有当 m n log n（其中 n 是顶点数，m 是输入图中的边数）时，才有可能使用最佳数量的处理器在对数时间内解决上述每个问题。我们的离线 LCA 计算使我们能够将这些问题的最佳加速对数时间并行算法范围扩展到更稀疏的图，其中 *m -^ n* log* *n* 与上述连接性算法相同。(3) 近似字符串匹配[6]。[7]中新的并行后缀树构造与目前的并行 LCA 计算相结合，大大简化了[6]中的并行算法。这种

简化在 [2] 中已有描述。

本文的结构如下。第 2 节给出了算法的高级描述。第 3 节介绍预处理阶段。在第 4 节中，我们展示了如何**利用**预处理阶段的结果在 T 中处理 LCA 查询。第 5 节介绍了预处理阶 段 的并行化。

2. 高级描述整个算法基于以下两个观察结果（也是在 [5] 中提出的）：(1) 如果我们的输入树是一条简单的路径，就有可能对其进行预处理（通过计算每个顶点到根的距离，如下文所述），然后在恒定时间内回答每个 LCA 查询。
(2) 如果我们的输入树是一棵完整的二叉树，就有可能对其进行预处理（通过计算其无序数，如下文所述），然后在恒定时间内回答每个 LCA 查询。

预处理阶段为 T 中的每个顶点 u 分配一个编号 I NLABEL (u)。根据观察结果 (1)，这些编号满足以下路径*分区属性：* I NLABEL 数字将树 T 划分为路径，称为 I NLABEL 路径。每条 I NLABEL 路径由具有相同 I NLABEL 数字的顶点组成。

让 *B* 成为至少有 n 个顶点的最小完整二叉树。我们的描述是用 *B* 中每个顶点的阶内编号来标识它。受观察结果（2）的启发，INLABEL 数字还满足以下 "*降序保留属性*"：INLABEL 数字将 T 中的每个顶点 u 映射到 *B* 中的顶点 I NLABEL (r)，这样， c 的后代就被映射到 *B* 中 INLABEL (c) 的后代（r 既被视为其后代，也被视为其祖先）。

考虑 T 中的顶点 u。根据后裔保留性质，r 的所有祖先都映射成 I NLABEL (c) 的祖先。这意味着，在 I NLABEL (c) 的所有祖先的 IN LAB EL 数字中，最多有 log n 个不同的数字。
u.稍后，我们将展示如何使用一个对数 n 位的字符串记录所有这些 INLABEL 数字。在预处理阶段，我们会将 T 中每个顶点 c 的字符串计算到 ASCENDANT (c) 中。

在预处理阶段，我们还要计算表 HEAD。它包含每条 I NLABEL 路径中最高的顶点。

第 4 节介绍如何处理 T 中任意一对顶点 *x*、*y* 的查询 LCA *(x，y)*。比较简单的情况是 x 和 *y* 属于同一条 I NLABEL 路径。在预处理阶段，我们会计算 T 中每个顶点 c 到 LEVEL (r) 的距离。因此，LCA (x, *y)* 只是 x 和 *y* 中更靠近根的顶点。更复杂的情况是 INLABEL (x) I NLABEL *(y)*。我们分四个步骤进行。第一步，在完整的二叉树 *B* 中找到 I NLABEL (x) 和 I NLABEL *(y)* 的 LCA，表示为
h.第二步，我们找到 IN LABEL (z*)*。I NLABEL（z）是 *B* 中 *b* 的最低祖先，也是 T 中 x 和 y 的共同祖先的 I NLABEL 编号。为此，我们使用 ASCENDANT *（x）*和 ASCENDANT *（y）*提供的信息。第三步和第四步，我们在 I NLABEL（Z）定义的 I NLABEL 路径中找到 z。在第三步中，我们在 T 中的 INI.ABEL (Z) 所定义的路径中找到 x 的最低祖先（记为 x）和 *y* 的最低祖先（记为 *y*）。考虑 *B* 中从 I NLABEL (z) 到 INLABEL (x) 的路径。我们从 ASCENDANT (x) 得出 x 在这条路

径中的祖先的第一个 I NLABEL 编号（即 *B* 的顶点）。表 HEAD 给出了 *x* 在 T 中具有该 I NLAB EL 编号的最高祖先。最后，x 是该祖先在 T 中的父节点。同样我们找到 *y*。第四步，我们找到 *z*，即 *k* 和 *y* 中更接近根的顶点。

3. 预处理阶段预处理阶段的结果包括分配给 T 的顶点的标签和一个名为 **HEAD** 的查找表。T 中每个顶点 u 的标签由三个数字组成：I NLABEL (c)、ASCEN- DANT (c) 和 LEVEL (u)。

我们首先计算 T 中每个顶点 u 的 INLABEL (v)。在讨论这两个步骤后，我们将展示如何实现它们。

设 PREORDER (u) 是 T 的预序遍历中 r 的序号，SIZE (t') 是以 r 为根的子树上的顶点数。

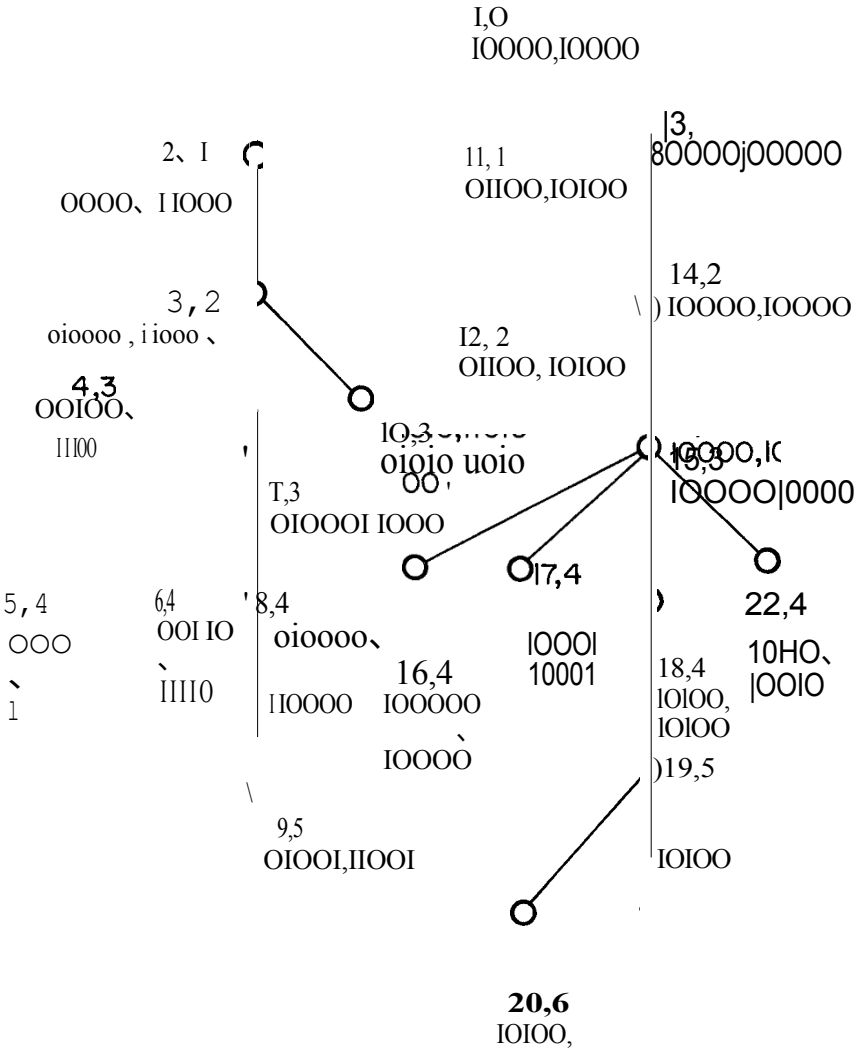
步骤 1.计算 PREORDER (u) 和 SIZE (r)。

我们注意到，以 u 为根的子树中各顶点的 PREORDER 数介于 PREORDER (r) 和 PREORDER (r) + SIZE (r) - 1 之间，因此封闭区间 [PREORDER (u), PREORDER (u) + SIZE (r) - 1] 称为 *v 的内部*区间。

在步骤 2 中，我们考虑 u 的区间内（整数）数的二进制表示。我们要说明的是，在本文中，我们交替提及数及其二进制表示。不会产生混淆。

步骤 2.找出在 r 的区间内最右边的 "0 "比特数最多的（整数）数字。

本节所述计算示例见图 3.1。



Fmi.3.1. *举例说明。一棵树上有四个数字：*PR EORD ER, LEVEL, IN LAB EL, mid ASCENDAN "f *在每个顶点。最后两个数字是* git'eri *iii 二进制表示）。*

讨论我们证明，I NLAB EL 数字满足上一节高层描述中定义的两个属性。

定理 1. INLAB EL *数字满足路径分区属性。*

证明请注意，r 的子区间必须是成对不相交的。因此，I NLABEL (v) 最多属于 r 的一个子项的区间。根据 I N LABEL 数字的选择（第 2 步），I NLAB EL (ii) = INLABEL (u)（如果 u 存在），对于 u 的任何其他子 w，I NLAB EL (w) N INLABEL (r)。这意味着 INLAB EL 数字

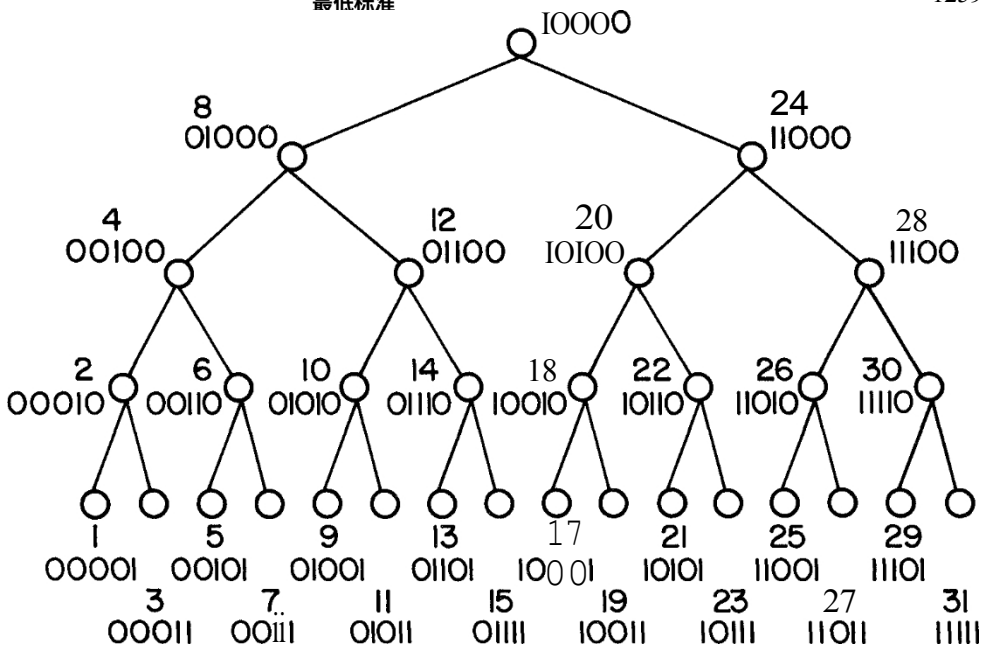
的*路径分区属性*。INLAB EL *数字满足降序保留属性。*

*证明*我们证明 I NLAB EL *(d*) 是完整二叉树 *B* 中 I NLAB EL (r) 的后代（回顾一下，我们的描述是用 *B* 中每个顶点的序号来标识它的，因此证明了这一 Lemma）。考虑 B 中的任意两个顶点 b 和 c。我们首先给出 c 是 *B* 中 *b* 的后代的必要条件和充分条件，然后证明 I NLABEL *(d)* 和 I NLABEL (u) 满足这个条件。设 I - [log *n)* 和 *i* 是 *b* 中最右边的 "0 "位的个数。也就是说，*b* 由 *l - i* 个最左边的位组成，后面跟着一个 "1 "和 *i* 个"0"。

说法*顶点 c 是 b 的后缀，当且仅当 (1) c 的最左端 l - i 位与 h 的最左端 l - i 位相同，且* (2) c *中最右端* "0 " *位的个数至多为 i。*

*证明*设 *bz* 和 *bp* 分别是 *b* 的左子和右子。不难看出以下几点：(i) be 由 h 的 *I - i* 个最左边的比特组成，后面跟一个 "0"、一个 "1 " 和 i - 1 个 "0"；(ii) *bz* 由 *b* 的 *- i* 个最左边的比特组成，后面跟两个 "1 "和 i - 1 个 "0"。这些事实很容易暗示我们主张的两个方向。

完整二叉树及其无序编号的示例见图 3.2。



音格3.2.*例题有3 1 个顶点的完整二叉树的无序编号。(数字也以二进制表示）。*

本文中所有对数的基数都是 2。

我们回到定理 2 的证明。设 *i* 是 INLABEL (u) 中最右边 "0 "位的个数。由于 INLABEL (d) 属于 r 的区间，而 INLABEL (u) 在这个区间中最右边的 "0 "比特数最大，所以 I NLABEL *(d)* 中最右边的 "0 "比特数最多为 *i*。特别是，I NLABEL *(d)* 中最左边的 *l* - *i* 比特与 I NLABEL (u) 中最左边的 - i 比特相同。这意味着 I NLAB EL *(d)* 是 INLABEL (u) 在 *B* 中的后代。O

执行。给定 PREORDER (u) 和 SIZE (c)，对于 T 中的每个顶点 r，步骤 (2) 在每个顶点的恒定时间内分两个子步骤实现。

步骤 2.1.计算 [log [(PREORDER (c) - 1) xor (PREORDER (r) + SIZE (u) - 1)]]到 *i*。让我们来解释一下。PREORDER (r) - 1 和 PREO RD ER (r) + SIZE (r) - 1 的位逻辑排他 OR（表示 xor）给 PREORDER (u) - 1 和 PREORDER (u) + SIZE (r) - 1 的每个位赋值 "1"。二进制）对数的底数给出了差值最左边位的索引（从索引为 0 的最右边位开始计算）。请注意，在 PREORDER (c) - 1 中，位索引 *i* 必须为 "0"，而在 PREORDER (r) + SIZE (u) - 1 中，位索引 *i* 必须为 "1"，因为第二个数字更大。

步骤 2.2 展示了如何 "组成 "INLABEL (r)。为此，我们需要两个观察结果。(1) I NLAB EL (c) 最左边的 *l* - *i* + 1 位与 PREORDER (r) + SIZE (r) - 1 最左边的 1 - *i* + 1 位相同。 (2) I NLABEL (c) 中的其他 *i* 位为 "0"。*步骤* 2.2.计算 2' [(PREORDER (u) + SIZE (u) - 1)/2'] 到 I NLABEL (r)。这样就把 PREORDER (r) + SIZE (u) - 1 中最左边的 *l* - i + 1 位分配给 I NLABEL (c) 中最左边的 / - i + 1 位，而把 "0 "分配给 I NLABEL (v) 中的其他位。

*备注*上述计算基于对 T 的顶点进行 PREORDER 编号。这种编号方法的特性是，分配给 T 的任意顶点为根的子树的编号都是连续的整数序列。事实上，任何其他具有这种特性的编号方式（如 POSTORDER、I NORDER）都会产生 INLABEI 数字，适合我们的预处理阶段。

我们接着计算 ASCENDANT 数字。总体思路是，对于每个顶点 r，单个数字 ASCENDANT (r) 将记录 T 中 u *所有*祖先的 I NLABEL 数字。我们注意到，从顶点 u 的角度来看，其每个祖先的 INLABEL 数字可以完全由其最右边的 "1 "的索引指定。这是因为这个 "1 "左边的位与 I NLAB EL (r) 中各自的位相同。与 INLABEL 数一样，ASCEN- DANT (r) 也是一个 *(I* + 1) 位数。用二进制序列 *A (o),-*, At(r*)* *表示* ASCEN- DANT (c) 的二进制表示。只有当 *i* 是 T 中 u 的祖先的 IN LABEL 数字中最右边的 "1 "的索引时，我们才将 A,(c) 设为 1。我们从

ASCENDANT (r) = 2' 开始。如果 IN LAB EL (c*)* - INLABEL (F(u))，那么我们将 ASCENDANT (F(u)) 分配给 ASCENDANT (o)；反之，我们将 ASCENDANT (F(*r))*+ 2' 到 ASCENDANT (c)，其中 i 是 I NLAB EL (u) 中最右边 "1 "的索引。可以很容易地验证，i 由 log（I NLABEL (r) - [I NLABEL (r) and (I NLABEL (r) - 1)]）给出，其中 和 表示比特逻辑 AND。

回顾一下，对于 T 中的每个顶点 c，LEVEL (u) 是指从 r 到根 r 的路径距离（算上边沿）。LEVEL 数字的计算非常简单，可以使用 "广度优先搜索 "等方法。

记得图 3.1 给出了一个标签示例。

最后，我们将介绍如何计算表 HEAD。HEAD (*k*) 包含由 INLABEL 数字为 k 的所有顶点组成的路径中最靠近根的*顶点*。表 HEAD 的计算非常简单。对于每个顶点 u，如果 INLABEL (u) INLABEL *(F(r))*，我们就将 r 分配给 HEAD (I NLABEL (c))。这同样需要线性时间和线性空间。

*一般实施说明。*预处理阶段和查询处理的时间限制取决于在恒定时间内执行乘法、除法、二幂运算、位和运算、基二离散对数运算和位排他或运算的能力。如果机器不具备这些运算能力，那么在预处理阶段，就需要在线性时间和线性空间内为每种缺失运算准备查找表。这些查找表将用于以 O(1) 的运 算 量 执行所缺运算。

4. 处理 LCA 查询。在本节**中**，我们将展示如何回答 LCA 查询**使用**预处理阶段的结果。

对于 T 中的任意一对顶点 x、*y*，考虑一个查询 LCA (x，y*)*（为便于说明，读者可参阅图 3.1）。

情况 A. IN LABEL *(x)* = I NLABEL *(y)*。*x* 和 *y* 肯定在同一条 INLABEL 路径上。我们的结论是，如果 LEVEL (x) LEYEL (y)，则 LCA (x, *y)* 为 *x*，否则为 *y*。

情况 B. I NLABEL (x) /1 NLABEL (y)。设 z 为 LCA（x，*y*）。我们分四步找到 z：

步骤 1.找到完整二叉树 *B* 中 INLABEL *(x)* 和 I NLABEL *(y)* 的 LCA *b*，如下所示。因为 *b* 是 *B* 中 I NLABEL (x) 和 I NLABEL *(y)* 的共同祖先，*所以 i* 必须满足以下两个条件。(1) INLABEL (x) 和 I NLABEL (y) 中最左边的 I - i 位与 *b 中*的这些位相同。 (2) I NLABEL (x) 和 I NLABEL (*y)* 中最右边的 "1 "的索引*最多为 i*。我们区分三种*情况。*

情况 (1)： I NLABEL (x) 是 I NLABEL (y) 的祖先。I NLABEL (x) 是 I NLABEL *(y)* 的祖先。设 i 是 I NLABEL *(x)* 中最右边的 "1 "的索引。注意，在这种情况下，INLABEL (x) 和 I NLABEL *(y)* 中最左边的 *l - i* 位是相同的，并且 I NLABEL (1) 中最右边的 "1 "的索引 < i, 。因此，*i* 等于 *i,* .

情况 (2)： I NLABEL (y) 是 INLABEL (x) 的祖先。I NLABEL *(y)* 是 INLABEL (x) 的祖先。与情况 (1) 类似，i 是 INLABEL *(y)* 中最右边的 "1 "的索引。

案例(3)。与情况（1）和（2）不同。在这种情况下，*i* 是***最小***索引，使得 IN LABEL (x) 和 I NLABEL *(y)* 中最左边的 *l* - i 比特相同。

我们只需将 *i* 取为以下各项中的最大值，就可以同时处理这三种情况：I

NLABEL (x) 和 I NLABEL *(y)* 分歧的最左边位的索引；I NLABEL (x) 中最右边 "1 " 的索引；以及 I NLABEL (*y)* 中最右边 "1 " 的索引。*b* 由 INLABEL (*x)* （或 I NLABEL (J)）中 最左边的 *l - i* 个位组成，后面跟一个 "1 "和 *i* 个"0"。

在步骤 2 中，我们找到 INLAB EL (r)（其中 z 是 LCA (x，*y)*）。I NLABEL 数字的后裔保留属性意味着 I NLABEL (z) 是 I NLABEL (x) 和 I NLABEL *(y)* 的共同祖先。请注意，I NLABEL (z) 不一定是 I NLABEL (x) 和 I NLABEL *(y)* 的*最低*共同祖先 *b*。这是因为 T 中映射到 *b* 的顶点不一定是 *x* 的祖先。

然而，不难看出，lN LABEL (z) 是 *b*

是 *x* 和 *y* 在 T 中的祖先的 INLABEL 号码。

步骤 2.查找 I NLABEL (z)。因为 r 是 *x* 和 *y* 在 T 中的共同祖先，所以 A,(x) = 1，A*;(y)* = 1。由于 INLABEL (z) 是 h 的*最低*祖先，是 *x* 和 *y* 的共同祖先，*因此*索引 *j* 必须是 *A x), - - , A,(x)* 和 *A (y)*-,- - , A,*(y)* 中*最右边的* "1 "的索引。IN LABEL (z) 由 I NLABEL (*x)* （或 I NLABEL *(y)*）最左边的 *I* -j 个位组成，后面跟一个 "1 "和 j 个 "0"。

为此，我们要找到 I NLABEL (*z*) 所定义路径中 *x* 的最低祖先 I，以及该路径中 *y* 的最低祖先 *y*。

步骤 3.求 *k* 和 *y*。我们将演示如何求 *k*。如果 INLABEL (x) = INLABEL (z)，那么 x = x，不需要做任何事情。假设 INLABEL (x) I NLABEL (r)。我们设定以下中间目标，作为找到 x 的主要步骤：找到 x 的子代，同时也是 x 的祖先。将我们搜索的顶点记为 w，让 *k* 成为 I NLABEL (*w)* 中最右边的 "1 " 的索引。不难验证，k 是 *A (x), ...*显然，INLABEL（ir）由 I NLAB EL（x）最左边的 f *k* 个比特组成，后面跟着一个 "1 " 和 *k* 个"0"。请注意，w 是其 IN LABEL 路径的首部（因为*其父* k 的 I NLABEL 编号与 I NLABEL (w) 不同）。因此，w 是 HEAD (INLABEL (w))，我们的中间目标就达到了。最后，x 是 w 的父节点。

步骤 4.如果 LEVEL(x) 等于 *k*，*则* LCA(x, y) 等于 y。 LEVEL(y)，否则为 *y*。

在本节的其余部分，我们将介绍上述处理过程所需的 其他实施细节。

步骤 1.要找到 *i*，*即 b* 中最右边的 "1 "的索引，我们要做如下操作。

步骤 1.1.求 *i,* ，I NLABEL (x) 中最右边的 "1 "的索引，以及 ' ，I NLABEL *(y)* 中最右边的 "1 "的索引。要找到 *i*，我们要计算 i , .= log (I NLABEL (x) - [I NLABEL (x) and (I NLABEL (x) - 1)]), 就像上一节的 ASCENDANT 数字计算一样。

步骤 1.2.找出 *i,* ，即 I NLABEL (x) 和 IN LABEL *(y)* 的最左边位的索引。 的最左边位的索引。为了找到 *i*，我们计算 i3 [log [I NLABEL (x) xor I NLABEL *(y)])* 。这与步骤 2.1 在上一节的 IN LABEL 号码计算中。

i 是 it , in 和 *i,* 之间的最大值。给定 *i 后*，*b* 的计算方法与 I NLABEL 数字计算中的步骤 2.2 类似。

步骤 2.要找到 j，我们需要执行以下步骤。

步骤 2.1.将 ASCENDANT (x) 和 ASCENDANT *(y)* 的位逻辑 AND 计算到 COMMON 中。

步骤 2.2.计算 2' [COMMON/2'] 到 COMMON, 中。COMMON，列表 *A x)-, -- , A,'x)* 和 *A y)*-, - - , A,*(y)* 中的所有 "1"。

y 是 COMMON, 中最右边的 "1 "的索引。为了求出 j，我们计算 j:= log（COMMON, - [COMMON; and (COM- MON, - 1)] ），就像上一节的 ASCENDANT 数计算一样。

步骤 3 的实施采用了相同的技术。

5. 并行预处理算法。在本节**中**，我们将介绍预处理阶段的并行版本。它使用 n/log n 个处理器，运行时间为 O(log n)。

我们对输入树 T 的表示作了如下假设：树 T 的 n - 1 条边是在一个数组中给出的，其中每个顶点的输入边是连续分组的。根据我们对树 T 的定义，它的边是指向根的。

*并行计算标签*为了计算 *T* 中顶点的标签，我们采用了计算树函数的欧拉游技术，该技术在 [10] 和 [12] 中给出。不过，我们将使用 [3] 的 O(log n) 时间最优并行列表排序算法来实现它。这种列表排序算法是为 EREW PRAM 设计的。它以扩展图为基础，其 O(log n) 时间约束隐藏了一个不大不小的常数。我们注意到，[4] 最近给出了另一种列表排序算法，其时间和处理器效率相同。这种替代算法是针对允许同时读写同一内存位置的 PRAM（称为 CRCW PRAM）设计的。它更简单，其 O(log n) 时间约束只需要一个很小的常数。

下面，我们首先回顾欧拉游技术所需的结构。然后，我们将展示如何使用它来计算标签。我们不得不重新介绍欧拉巡回技术的唯一原因是，ASCENDANT 数字的计算在其他地方没有出现过。

步骤 1.对于 T 中的每一条边 (u -' ii)，我们添加它的反平行边 (u -r u)。让 If 表示新的图。

由于 *H* 中每个顶点的indegree 和 outdegree 都相同，因此 *H* 有一条以 r 为起点和终点的欧拉路径。第 2 步将这条路径计算到指针向量 *D 中，*对于 *H 中*的每条边 e，*D(e) 都*有欧拉路径中 e 的后继边。

步骤 2.对于 H 的每个顶点 r，我们进行如下操作：(让 u 的出边为 (r -r up)-, - - , (r - r uq ,).*D(up) '>(*i-4I-*) mod d),* for *i* = 0-, - - , *d* - 1. 现在 *D* 有一个欧拉回路。修正 "D(ripe, -r r) := *list end-of-list*（其中 r 的outdegree 为 *d*）给出了一条以 r 为起点和终点的欧拉路径。

我们将展示如何使用欧拉路径为 T 中的每个顶点 c 找出 PREORDER (r)、PREOR- DER (u) + SIZE (u) - 1 和 LEVEL (u)。

步骤 3.我们分配两个权重：*Wk(e)* 和 (e) 给欧拉路径中的每条边 *e*，具体如下。(1) 如果 e 的方向来自 r（即 *e* 不是树边），则 *Wk(e)* - 1，否则 IV,(e) = 0。(2) *(e)* = 1（如果 e 的指向来自 r），否则 *(e)* = -1，否则。

步骤 4.我们应用两次最佳对数时间并行列表排序算法，找出 *H* 中每个 *e* 与欧拉路径*起点*的（加权）距离：第一次应用相对于权重 W，结果存储在 DISTANCE, *(e)中；*第二次应用相对于权重 IV，结果存储在 DISTANCE (e)中。考虑一个顶点 r r，让 u 成为它在 T 中的父顶点。PREORDER (r) 是 DISTANCE, (u o r) + 1，PREORDER (o) + SIZE (u) - 1 是 DISTANCE, (r -r ii) + 1、

而 LEVEL (u) 就是 DISTANCE2(" o)。(读者可以很容易地验证这些说法）。

步骤 S. 给定 PREORDER (r) 和 PREORDER (u) + SIZE (u) - 1，对于 T 中的每个 ver- tex r，我们使用 n 个处理器在恒定时间内计算 I NLAB EL (r)，与串行算法一样。

接下来，我们将展示如何使用欧拉路径为 *T* 中的每个顶点 r 找出 ASCENDANT (r)。

步骤 6.我们为欧拉路径中的每条边 *e* 分配一个（新的）权重 W(e)，如下所示。对于每个顶点 u r 的操作如下。让 ii 成为 u 在 T 中的父节点，让 i 成为 I NLAB EL (v) 中最右边的 "1 "的索引。如果 I NLAB EL (o) INLABEL (ii)，我们指定 W(ii -r r) = 2' 和 IV(u o u) = -2'。所有其他边的权重设为零。

第 7 步我们再次使用并行列表排序算法，找出 H 中每个 e 与欧拉路径起点的（加权）距离。考虑一个顶点 r r，让 u 成为它在 T 中的父顶点。ASCENDANT (r) 是边 (u -r v) 的距离加上 2'。显然，ASCENDANT (r) - 2'。

我们注意到，在给定标签的情况下，可以使用 n 个处理器在恒定时间内计算出表格 HEAD。

步骤 4 和步骤 7 各需要 n/log n 个处理器和 O(log n) 时间。步骤 1、2、3、5、6 中的每一步以及 HEAD 的计算都需要 n 个处理器和 O(1)个时间，并可由 n/log n 个处理器在 O(log n) 个时间内轻松模拟。因此，并行预处理阶段可以使用 $n/\log n$ 个处理器在总计 O(log n) 时间内完成。

致谢。我们感谢 Noga Alon 和 Yael Maon 激发了我们的讨论。我们还要感谢匿名审稿人提请我们注意 [9]。

RE FERE NCES

[1] A.V. Ano, J. E. Her xc RO rT, AN D *I.* D. U LLMAN, *The **Design and Analysis** of Computer **Algorithms***、马萨诸塞州雷丁市 Addison-Wesley，1974 年。

[2] A.APOSTo Pico, C. Imo ro u nos, G. M. LAN DAY, B. Sc u iEB ER, AN O U. ViSH KI N, *Parallel construc- tion of a suffix tree with applications,* Algorithmica Special Issue on Parallel and Distributed Computing, t o appear.

[3] R.CoLE **AND** U. VMSfi KI N, *Approximate **and** exar.t **f'arallel scheduling** with **applications** Io list, tree **and** graph problems,* in Proc. 27th Annual IEEE Symposium on Foundations of Computer Science, 1986, pp.

[4] ———莫伊兹和弗里达-埃斯肯努西的 "*平行前缀总和与列表排序优化大师*"，TR 56/86 特拉维夫大学计算机科学研究所，以色列特拉维夫，1986 年。

[5] D.HARE L AND R. E. TA RJAN, *FQSt Algorithms for finding nearest common ancestors,* SIAM J. Comput., 13 (1984), pp.

[6] G. *M.* LAN DAU AN D U. VISH Kl N, *Introducing efficient parallelism into apf'roximate .string matching*、第 18 届 ACM 计算理论年度研讨会论文集》，1986 年，第 220-230 页。

[7] G. M. LANDAU, B. Sc H IEB F-R, AND U. VIs fa KI N, *Parallel construction of a sufjix tree,* in Proc. 14th lnternat.Colloquium on Automata Language and Programming, Lecture Notes in Computer Science 267, Springer-Verlag, Berlin, New York, 1987, pp.

(8]Y. MAO N, B. Sc ri iEB ER, A ND U. VISH KIN, *Parallel ear decomposition search* (EDS) *and .st-numbering.* in graphs, Theoret.Comput.Sci., 47 (1986), pp.

[9] Y.H. TSI N, *Finding lowest common ancestors in parallel,* IEEE Trans.计算》，35 (1986)，第 764-769 页。

[10] R.E. TARJAN AN O U. Vi SH KI N, *An* e#icient *parallel biconnectivit y algorith m,* SIAM J. Comput. (1985), pp.

(11]U. VISH KIN, *Synchronous fiarallel computation -a surue y,* TR-71, Department of Computer Science、纽约大学库兰特数学科学研究所，纽约州纽约市，1983 年。

——— [12] *On efficient parallel strong orientation,* Inform.Lett.Lett., 20 (1985), pp.