海量数据的可扩展K阶最长公共前缀数组构造方法

吴裔、农革、刘伟军

(中山大学信息科学与技术学院 广州 510006)

Scalable K-Order LCP Array Construction for Massive Data

Yi Wu, Ge Nong and Wei Jun Liu

(Department of Computer Science and Technology, Sun Yat-Sen University, Guangzhou, China)

摘 要: 本文提出一种使用后缀数组（SA）构造海量数据K阶最长公共前缀（LCP）数组的方法。该方法限定输入字符串T中任意两个后缀的最长LCP为K个字符，并使用一个指纹函数将LCP构造过程中的字符串比较运算转化为可在常数时间内完成的整数比较运算来降低时间复杂度。所述方法适用于内存、外存和分布模型，在各模型上的总时间和空间复杂度均为和。其中，在由d个计算节点构成的分布模型上，各节点的时间和空间复杂度分别为和。

关键词: 后缀数组；K阶最长公共前缀数组；指纹函数；海量数据

**Abstract**: A new method is proposed to compute the K-order longest common prefix array via the suffix array for a size-n input string T, where the maximum LCP of a pair of suffixes of T is truncated to be at most K characters. By employing a finger-printing function, string comparisons in this method are translated into integer comparisons. This method can be applied on the internal memory, external memory and distributed computation models. For each of these models, the total time and space complexities are O(n logK) and O(n), respectively. In particular, this method is scalable for a distributed model of d computing nodes, where the time and space complexities are evenly divided onto each node as O(nlogK/d) and O(n/d), respectively.

**Key words**: suffix array; K-order longest common prefix array; finger-print function; massive data

# 导言

大规模随机二进制数据的模式查找技术是盲信号处理领域的一个研究热点，由于随机二进制数据缺乏语法、语义特征，无法应用分词等预处理手段来建立哈希索引，因此需要构造和使用全文索引来提升模式查找效率。后缀数组[1]是一种常用的全文索引数据结构，其结构紧凑，可用于分析、提取盲信号的数据特征。SA加上以其为基础计算的最长公共前缀数组（LCPA），只需不到一半的空间就可用同等的时间实现后缀树的遍历、查找等操作[2]。

在内存模型上，目前所知性能最优的SA和LCPA构造算法采用归纳排序（IS）方法[3][4]。但随着数据采样技术的进步，现有内存构造算法已难以处理规模持续增长的待处理数据。近期出现的几个外存算法[5][6][7]以不同方式将内存模型上的IS方法扩展到外存模型并取得了远优于其他外存算法[8]的时空效率。其中，eSAIS算法[6]可在计算SA的过程中同步计算LCPA，时间和I/O开销是单独计算SA的两倍。eGSA算法[9]采用多路归并排序方法同步计算字符串集合的全局SA和LCPA，所需时间为eSAIS算法的三分之一。算法exLCP[10]是一种轻权LCPA构造算法，其计算字符串集合的Burrow-Wheeler变换来构造LCPA。LCPscan[11]算法根据SA来间接构造LCPA。与eSAIS算法相比，LCPscan算法的磁盘工作空间较小且在运行时间和I/O效率上的性能更好。上述算法在时空性能上的性能表现良好，然而它们的设计复杂、扩展性差，难以在并行和分布模型上利用多个计算节点来提升性能。

另一方面，统计数据表明实际数据（尤其是蛋白质和DNA等基因数据）的平均LCP长度通常较小[9]。这促使我们设计实用算法来计算输入字符串T的K阶LCPA，算法假设T的任意两个后缀的最大LCP不超过K且K远小于T的长度（例如：K=8192）。

本文的贡献主要包括以下两个方面：

1. 我们的第一个贡献是设计和实现一个K阶LCPA构造算法。算法采用LCP批量查询技术在O(nlogK)的时间和O(n)的空间内构造输入字符串的LCPA，适用于内存和外存模型，编写的C++程序代码少于600行。
2. 现存并行构造算法主要用于整体同步并行和并行随机访问等共享内存计算模型[13][14]。我们的第二个贡献是将所提出的方法移植到由d个计算节点组成的传统分布系统上。

本文的其余部分组织如下。第2节介绍LCP-BQT技术并描述所提出的构造方法在内存模型上的算法框架。第3节将所述方法扩展到外存和分布模型上。最后，我们分别在第4和第5节给出性能分析和总结。

# 内存上构造K阶最长公共前缀数组

## 名词解释

考虑有序字符集上长度为n的输入文本。假设是上的最小字符且只在T中出现一次。为便于描述，我们引入以下名词解释。

* 分别将由T[0]T[1]…T[i]和T[i][i+1]…T[n-1]构成的前缀和后缀记作和。
* 将T的后缀数组记作。是区间中各整数的一个全排列，其满足字典次序上的小于关系：
* 将后缀和的最长公共前缀长度记作。最长公共前缀数组是由取自区间的n个整数构成的，其满足。
* 将记作。

## LCP批量查询技术

已知字符串T和b对索引构成的集合P，LCP-BQT可使用O(nlogb)的时间和O(n)的内存空间来批量计算P中各对索引的最长公共前缀长度。主要思路是寻找使得对有和成立。为此，其首先令，然后执行logb次循环。其中，第k次循环的目标是确认中的各对是否满足并按如下方式生成：若，则将插入中；否则将插入中。上述过程中涉及的字符串比较可依据从左向右按位比较字符的方式来求解。从而，整个过程的最坏时间复杂度为，容易成为性能瓶颈。

一个可行的解决方案是采用文献[15]中的指纹函数将字符串比较转换为可在常数时间内完成的整数比较。字符串的指纹可由公式计算得到。其中，L是与一个质数，而是一个从区间[1,L)中随机提取的整数。显然，两个相同字符串的指纹相同，但反之不成立。幸运的是，现已证明当L足够大时，两个不同字符串具有相同指纹的概率可忽略不计。

根据以上描述，我们给出第k轮循环的三步算法框架：

1. 从右向左扫描字符串T并根据公式来迭代计算的指纹。若，则在哈希表中存储。
2. 对每个，根据公式来计算的指纹。
3. 对每对，比较与的大小。若相等，则将插入中；否则将插入中。

整个循环过程满足以下两个不变式：

1. 在第k轮循环的开始，对每对，有成立。
2. 在第k轮循环的结束，对每对，有成立。

在整个循环结束后，对每对，有成立，从而我们能够在的时间内从左向右按位比较和的大小。令且，则和分别是位于和右侧的位置索引且满足和。

引理**1**. 可在O(nlogb)的时间和O(b)的空间内以极大概率正确计算T中任意b对后缀的LCP。

证明: LCP-BQT的时间复杂度取决于logb轮循环，其中每轮循环在步骤S1使用O(n)的时间迭代计算T中各前缀的指纹，并在步骤S2-S3使用O(b)的时间来计算和比较字符串的指纹。此外，算法使用O(b)的空间来使用哈希表存储和获取迭代计算的指纹。

## 算法描述

将LCP数组构造问题转化为批量求解集合中各对索引的LCP。为便于说明，首先引入以下名词解释，其中且。

* 和是两个大小为2n的整数数组。
* 和分别为基数排序和得到的两个大小为2n的整数数组。
* 是存储和提取前缀指纹的哈希表。其中，。

如图1所示，算法1在初始阶段按如下步骤计算和：1）且；2）且。此后，算法在第4-9行循环计算和。在第k轮，算法迭代计算T中前缀的指纹并据此计算得到和。若两个指纹相等，则将和分别加上且和分别加上。算法将更新后的和分别赋值为和。在整个循环结束后，算法使用O(n)的时间计算目标LCP数组。从而，我们可得到以下结论：

引理**2**. 已知T的后缀数组，可使用O(nlogK)的时间和O(n)的空间以极大概率正确计算T的K阶LCP数组。

证明: while循环使用O(nlogK)的时间来计算和。另一方面，需要O(n)的内存空间来存储哈希表HT，和。

# 外存上构造K阶最长公共前缀数组

内存算法lcp-ram中使用一个哈希表来快速查找指定字符串的指纹。当T的长度n变大时，算法无法在内存中长期维持哈希表的一个完整副本。为此，我们重构lcp-ram来设计磁盘友好的外存算法lcpa-disk。该算法能够顺序地从外存中读取目标字符串的指纹，从而避免因随机I/O导致的系统性能下降。

## 预备知识

已知内存容量为M，我们在外存模型中将字符串和后缀数组均匀划分为的子块。其中，每块的大小为，从而可在内存中处理。我们将在算法lcpa-ram中使用的和扩展为和，两者均包含2n个子项，各子项的内容为一个的三元组，其中各数据域描述如下：

* 表示子项的索引，满足。
* 表示前缀pre(0,pos)的结束位置索引。
* 表示前缀pre(0,pos)的指纹。

对于，和的fp值被用于计算。类似可得的值。

## 算法描述

lcpa-disk算法对子项执行两此外存排序，排序关键字为logn比特的pos或idx。算法首先按pos将数组和排序来得到和，然后扫描和的子项来迭代计算所需的指纹。其后，算法按idx排序数组和来得到和。给定大小为的I/O缓冲区和内存空间，可采用一种多路基数排序方法在O(n)时空复杂度内执行所述外存排序。从而，我们得到以下结论：

引理**3**. 已知T的后缀数组，可使用O(nlogK)的时间和O(n)的磁盘空间以极大概率正确计算T的K阶LCP数组。

## 算法优化

为降低lcpa-disk中while循环的时间开销，可从和直接计算和，从而将循环次数减半。为此，将算法过程中的第8-9行修改如下：

1. 计算和比较和的指纹。若相等，则执行步骤S2，否则跳转执行步骤S3。
2. 计算和比较和的指纹。若相等，则令和自增并令和自增；否则，令和自增并令和自增。
3. 计算和比较和的指纹。若相等，则令和自增并令和自减；否则，令和自减+。
4. 令且。
5. 令。

## 讨论

在本小节，我们展示如何将外存lcpa-em算法扩展到包含d个计算节点的分布模型上。所有计算节点由一个工作在全双工模式下的千兆以太网交换机内连，且单个计算节点上的内存和外存空间分别为M和E。将字符串和后缀数组均匀地切分为块并将分块和分布到节点上，其中各块的大小。

算法lcpa-disk通过两次外存基数排序来计算和。在分布系统中，我们使用一组发送/接收缓冲区来实现计算节点间的数据交换从而模拟外存排序的过程。在while循环结束后，可直接由位于节点上的、和来计算得到的LCP数组。最后，我们可将各计算节点上的部分LCP数组前后相连得到全局解。

引理**4**. 已知T的后缀数组，可使用时间和单个计算节点上的磁盘空间以极大概率正确计算T的K阶LCP数组。

证明: 算法使用收发缓冲区在计算节点间模拟外存基数排序，在整个循环中用于传输数据的时空开销为O(elogK)。这是因为每个节点在每轮循环中发送和接收O(e)个子项。

# 实验结果

我们在一个实际数据集（http://download.wikimedia.org/enwiki）上测试算法lcpa-disk和lcpa-disk-m的性能。实验环境是一台配备有英特尔i5处理器、4GB内存和500GB硬盘的笔记本电脑。

我们使用STXXL[16]来实现程序中的外存排序过程。STXXL是一个旨在实现外存高效计算的C++STL库，最新版本可从http://stxxl.sourceforge.net上免费获取。借助STXXL提供的性能强大的优先权队列， lcpa-disk和lcp-disk-m算法的C++代码分别只有400和600行。

表1中的每个实验结果取自两次测试的平均值，其中的磁盘峰值和运行时间统计自stxxl::block\_manager和time命令。如表示所示，lcpa-disk-m的运行时间远快于lcpa-disk，其中两者的while循环次数分别是7和13。我们同时发现，随着测试集规模从200MB增加到2GB，每个字符在每轮循环的处理时间也从1.99增长到4.18。导致该非线性现象的部分原因在与STXXL采用非线性的外存排序算法来实现优先权队列，可采用本文所述的基数排序算法来实现线性增长。

文献[11]中的实验表明，eSAIS和LCPscan算法的磁盘工作空间分别为65n和21n，而lcpa-disk-m在测试集为2GB的下的磁盘工作空间为61n。这表明lcpa-disk-m在空间需求上的表现不及现有最佳算法。然而，本算法的优势在于实现简单且能够较好地扩展到并行和分布模型上，各节点的通信开销均衡且为O(n/d)。

# 总结

我们在本文中提出了一个实用的K阶LCPA构造方法，该方法适用于内存和外存模型。当使用STXXL来实现外存排序时，算法lcpa-disk-m的C++代码少于600行。我们展示了如何将所述方法直接扩展到由d个计算节点构成的经典分布模型上。一个由局域网内连而成多节点分布系统在实际中并不少见，但目前缺乏可在该分布系统上运行的可扩展LCPA构造算法，我们的工作有效填补了这一空白。作为后续工作，我们将尝试在该分布系统上实现所述方法并通过GPU来提升个模型上的算法性能。

参考文献

1. U. Manber and G. Myers. Suffix Arrays: A New Method for On-line String Searches. SIAM Journal on Computing, 22(5):935-948, 1993.
2. M. Abouelhodaa, S. Kurtzb, and E. Ohlebuscha. Replacing Suffix Trees with Enhanced Suffix Arrays. Journal of Discrete Algorithms, 2(1):53-86, November, 2004.
3. G. Nong, S. Zhang, and W. H. Chan. Two Efficient Algorithms for Linear Time Suffix Array Construction. IEEE Transactions on Computers, 60(10):1471-1484, October 2011.
4. J. Fischer. Inducing the LCP-Array. In Algorithms and Data Structures, volume 6844 of Lecture Notes in Computer Science, pages 374-385. 2011.
5. G. Nong, W. H. Chan, S. Q. Hu, and Y. Wu. Induced Sorting Suffixes in External Memory. ACM Transactions on Information Systems, 33(3):12:1{12:15, March 2015.
6. T. Bingmann, J. Fischer, and V. Osipov. Inducing Su\_x and LCP Arrays in External Memory. In Proceedings of the 15th Workshop on Algorithm Engineering and Experiments, pages 88-102.
7. G. Nong, W. H. Chan, S. Zhang, and X. F. Guan. Su\_x Array Construction in External Memory Using D-Critical Substrings. ACM Transactions on Information Systems, 32(1):1:1{1:15, January 2014.
8. R. Dementiev, J. Karkkainen, J. Mehnert, and P. Sanders. Better External Memory Suffix Array Construction. ACM Journal of Experimental Algorithmics, 12:3.4:1-3.4:24, August 2008.
9. F. Louza, G. Telles, and C. Ciferri. External Memory Generalized Su\_x and LCP Arrays Construction. In Proceedings of the 24th Annual Symposium on Combinatorial Pattern Matching, pages 201{210, Bad Herrenalb, Germany, June 2013.
10. M. Bauer, A. C. G. Rosone, and M. Sciortino. Lightweight LCP Construction for Next-Generation Sequencing Datasets. In Proceedings of the 12th International Workshop on Algorithms in Bioinformatics, Ljubljana, Slovenia, September 2012.
11. J. Karkkainen and D. Kempa. LCP Array Construction in External Memory. In Proceedings of the 13th International Symposium on Experimental Algorithms, pages 412{423, Copenhagen, Denmark, June 2014.
12. P. Bille, I. L. G\_rtz, T. Kopelowitz, B. Sach, and H. W. Vildh\_j. Sparse Suffix Tree Construction in Small Space. pages 148{159, July 2013.
13. J. Shun. Fast Parallel Computation of Longest Common Pre\_xes. In Proceedings of the 40th International Conference on High Performance Computing, Networking, Storage and Analysis, pages 387{398, New Orleans, LA, November 2014.
14. M. Deo and S. Keely. Parallel Su\_x Array and Least Common Pre\_x for the GPU. In Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pages 197{206, New York ,USA, August 2013.
15. R. Karp and M. Rabin. E\_cient Randomized Pattern Matching Algorithms. IBM Journal of Research and Development, 31(2):249{260, March 1987.
16. R. Dementiev, L. Kettner, and P. Sanders. STXXL: Standard Template Library for XXL Data Sets. Software: Practice and Experience, 38(6):589-637, 2008.