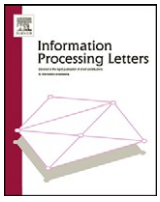
Information Processing Letters 110 (2010) 655-658

目录可在 ScienceDirect 上查阅

信息处理信函

www.elsevier.com/locate/ipl



几乎递增的最长子序列

Amr Elmasry [1]

德国萨尔布吕肯马克斯-普朗克信息研究所

Article info

文章历史：

2010 年 2 月 11 日收到

2010 年 5 月 24 日收到修订稿 2010 年 5 月 25 日接受

2010 年 6 月 1 日在线查阅 M. Yamashita 撰文

关键词：

算法

算法设计与分析 数据结构

ABSTRACT

给定一个包含 n 个元素的序列，我们引入了一个几乎递增子序列的概念，即通过给每个元素添加一个值（最多是一个固定常数），可以转换为递增子序列的最长子序列。我们展示了如何在 $O(n \log k)$ 时间内优化构造这种子序列，其中 k 是输出子序列的长度。

1. 导言

最长递增子序列（LIS）是长度最大的子序列，其中每个元素都大于前一个元素。最长递增子序列问题指的是产生子序列或仅仅找出其长度。70 年前，Robinson [16] 首次提出了这个问题。该问题的经典动态编程算法出现在许多算法教科书中[9]，是由申斯泰德[17]提出的。该算法的运行时间为 $O(n \log k)$，其中 n 是输入序列的长度，k 是最长递增子序列的长度。Knuth[14]给出了与 Young tableaux 有关的问题的生成式。Fredman [12] 证明，在比较树模型中，$O(n \log n)$的比较对于找出长度或产生子序列既是必要的，也是适当的。代数决策树模型也证明了同样的下限 [15]。如果输入序列是整数 1 到 n 的排列组合，那么引入的算法可以在 $O(n \log \log n)$ 时间内构造出最长的递增子序列 [8,13]。

这个问题在实践中非常重要。其他一些问题也涉及 LIS 结构（例如，见 [5]）。

电子邮件地址： elmasry@mpi-inf.mpg.de.

[1] 由亚历山大-冯-洪堡奖学金资助。

doi:10.1016/j.ipl.2010.05.022

最近，它在用于基因组比对的 MUMmer 系统[10]中得到了应用，因而变得更加重要。

一个相关的问题是最长公共子序列（LCS）问题，该问题考虑两个序列，并找出在两个序列中出现顺序相同的一系列条目。请注意，我们可以将 LCS 算法应用于序列及其排序结果，从而得到一个最长的递增子序列。

LIS 问题有 几种变体。循环列表的最长递增子序列（LICS）假定输入序列是循环的。循环列表最长递增子序列（LICS）的运行算法在预期的 $O(n\ \log\ n)$ 时间在 [3] 中给出。最佳最坏情况已知的边界为 $O(n^2)$，可以使用 [4] 中的技术实现。另一种变式是找到位于所有指定宽度滑动窗口中的最长递增子序列。文献[4]给出了一种算法，其运行时间与输出子序列的大小成正比，同时还给出了构建一个 LIS 的加法边界。文献[2]讨论了 LIS 问题的广义化，其中给出了一组固定的排列组合，任务是针对给定的输入序列，计算与给定排列组合之一阶同构的最长子序列。LIS 问题适用于此类排列为同序排列的情况。

这个问题的组合学也同样重 要 。从厄尔多斯和塞克雷斯的研究开始

的长度进行了研究。长度为 n 的均匀随机排列的 LIS 长度的完整极限分布在 [6] 中给出。结果表明，LIS 的预期长度接近 2’n。

假设我们正在考虑对以下过程进行监测活动的性能。一旦该活动在与大量经过认可的历史快照进行比较时表现良好，我们就认为该活动表现良好。如果该活动严格来说在之前选定的这些点中表现较好，但却选择了最大数量的点，这就过于局限和不公平了。这一概念需要放宽，以反映良好的进展，而不一定是迄今为止选出的最佳点。另一方面，在一些应用中，数据项存在少量噪声。因此，需要一个宽松版本的 LIS 问题。

在本文中，我们引入了 LIS 问题的一个变体我们称之为最长几乎递增子序列 （LaIS）问题。我们允许从目前出现的最大元素中最多减少一个常数值。给定一个序列 $(x_1, x_2, \ldots, x_n)$ 和一个常数 $c > 0$，我们的目标是构造一个最长子序列 $(y_1, y_2, \ldots, y_k)$ 这样，$6_i\, y_i > \max y^{i-1}_j - c$。我们给出一个渐近的最优算法，运行*时间*为 $O(n \log k)$。主要我们的想法是，将动态编程范式应用于利用指针进行搜索的数据结构，而不是数组。

2. 递归表述

让 $LaIS(h,i)$，$h \le i$，表示 $x_1, \ldots, x_i$ 的元素中几乎递增的最长子序列，使得最大元素是 x_h，且 h 最小。我们证明了这两个参数完全表征了任何 $LaIS$，并用较小 问题的解来重新表达 $LaIS$。

关键的一点是，任何 $LaIS(h,i)$，$h < i$，都可以按照关系拆分成两个独立的（h 值除外）子序列：

$$LaIS(h, i) = LaIS(h, h) - T(h, i)、$$

其中，"¬"是连接操作，$T(h, i)$ 是包括 $x_{h1}, \ldots, x_i$ 中满足 $x_h\, c < x_j \le x_h$ 的每个元素 x_j 的子序列。

第二个观察结果是，$LaIS（h，h）$ 可以表示为

$$LaIS(h, h) = LaIS\, i^{ir}, h - 1^{¢} - (x_h)、$$

其中 $length(LaIS(^{ir}, h\ 1))$ 的最大值为 r

最大元素为 x 的 子序列$_j$，且 j 为最小值。对于每个这样的子序列，只需跟踪其长度 l_j，以及 x 之前的元素中最大元素的最小索引 p_{jj}。在*第* i *次*迭代中，要执行两项任务：第一项任务是找到一个最长的几乎递增的子序列，其最大元素为 x_i。这个子序列的构建方法是，将 x_i 追加到迄今为止找到的最长的子序列*中，其最大元素为* x。$_i$更具体地说，我们在所有具有 $x_j < x_i$ 的索引 $j < i$ 中寻找一个索引 $^{ir} < i$，使得 l_i r ≥ l_j。然后将长度 l_i 设为 l_i r 1，并将索引 p_i 设为 ^{ir}。第二项任务是将 x_i 附加到迄今为止找到的最大元素大于或等于 x_i 且小于 $x_i\, c$ *的*每个子序列。更具体地说，对于所有 $j <$，如果 $x_i \le x_j < x_i\, c$，则将 l_j 设为 l_j 1。经过 n *次*迭代后，LaIS 的长度为最大长度 l_m

在算法存储的 l *个*$_i$ '中。为了构建 LaIS，我们利用 p_i 's 按相反顺序生成子序列。使用与最大长度 l_m 对应的元素 x_m，扫描从 $i\ n$ 到 m 1 的每个元素 x_i，并输出满足 $x_m\, c < x_i \le x_m$ 的元素，然后是 x_m。让 x_t 作为按相反顺序输出的子序列的最后一个元素，扫描从 $i = t$ - 1 到 p_t + 1 的每个元素 x_i，并输出满*足元素：* $i = t - 1$ *到* p + 1。满足 $x_{p_t} - c < x_i \le x_{p_t}$，然后是 x_{p_t}。前面的重复这一步骤，直到我们得到表示子序列第一个元素的 p_t 值（例如，当 $p_t\, t$ 时）。

前面算法的一个直接实现方法是使用两个链接列表（或两个数组），每个列表的大小都是 n；一个用于 l_i 's，另一个用于 p_i 's。这种实现方式的运行时间为 $O(n^2)$。

示例

假设 c 2，并考虑以下从 1 到 12 的序列：7、15、2、14、14、6、8、11、17、15、14、13。表 1 显示了算法进行 12 次迭代后的两个数组：一个数组保存长度 l_i 's（第一行），另一个数组保存索引 p_i 's（第二行）。

因此，LaIS 的长度为 6，这样的子序列有五个： 7, 15, 14, 14, 15, 14 , 7 , 6, 8, 11, 15, 14, 2, 6, 8, 11, 15, 14, 7, 6, 8, 11, 14, 13 和 2, 6, 8、11, 14, 13。通常，算法会存储一个 p_i，并重新导入一个 LaIS。

4.改进后的算法

我们并不存储与每个 x_i 对应的一个 l_i ，而是为每个长度值存储一个元素。也就是说，在*第* i *次*迭代后，我在 $i < h$ 之间，满足 x_i r < x_h 。注意，i 不一定是本质上是独一无二的。

3. 基本算法

算法经过 n 次迭代。第 i *次*迭代后，算法会在前 i *个*元素中为每个元素 x_j 保留一个最长的几乎递增的元素。

们存储长度为 l 的 z_l ▪ x_h ，其中 $length(LaIS(h, i)) = l$ 和 $x_h \le x_h$ r 适用于所有 hr sat-
满足 $length(LaIS(^{hr}, i)) = {}^{lr} \ge l$。由此可见，$z_j \le z\ r_j$
为了证明这些元素足以构造一个 LaIS，考虑两个元素 $x_h >$ x_h r，其中 $length(LaIS(^{hr}, i)) \ge length(LaIS(h, i))$。给定任何以 $LaIS(h, i)$ 为前缀子序列的几乎递增的子序列，我们可以用 $LaIS(^{hr}, i)$ 代替 $LaIS(h,i)$，得到另一个长度至少相同的几乎递增的子序列。

表 1

基本算法的模拟运行

迭代 1	**1** **1**										
迭代 2	1 1	**2** **1**									
迭代 3	1 1	2 1	**1** **3**								
迭代 4	1 1	**3** 1	1 3	**2** **1,3**							
迭代 5	1 1	**4** 1	1 3	**3** 1,3	**2** **1,3**						
迭代 6	**2** 1	4 1	1 3	3 1,3	2 1,3	**2** **3**					
迭代 7	2 1	4 1	1 3	3 1,3	2 1,3	2 3	**3** **1,6**				
迭代 8	2 1	4 1	1 3	3 1,3	2 1,3	2 3	3 1,6	**4** **7**			
迭代 9	2 1	4 1	1 3	3 1,3	2 1,3	2 3	3 1,6	4 7	**5** **2,8**		
迭代 10	2 1	**5** 1	1 3	3 1,3	2 1,3	2 3	3 1,6	4 7	5 2,8	**5** **8**	
迭代 11	2 1	**6** 1	1 3	**4** 1,3	**3** 1,3	2 3	3 1,6	4 7	5 2,8	**6** 8	**5** **8**
迭代 12	2 1	6 1	1 3	**5** 1,3	**4** 1,3	2 3	3 1,6	4 7	5 2,8	6 8	**6** 8	**5** **8**

表 2

改进算法的模拟运行。

(a) z_i 's 阵列								
迭代 1	7							
迭代 2	7	15						
迭代 3	2	15						
迭代 4	2	14	15					
迭代 5	2	14	14	15				
迭代 6	2	6	14	15				
迭代 7	2	6	8	15				
迭代 8	2	6	8	11				
迭代 9	2	6	8	11	17			
迭代 10	2	6	8	11	15			
迭代 11	2	6	8	11	14	15		
迭代 12	2	6	8	11	13	14		

(b) p_i 's 阵列											
迭代 1	1										
迭代 2	1	1									
迭代 3	1	1	3								
迭代 4	1	1	3	3							
迭代 5	1	1	3	3	3						
迭代 6	1	1	3	3	3	3					
迭代 7	1	1	3	3	3	3	6				
迭代 8	1	1	3	3	3	3	6	7			
迭代 9	1	1	3	3	3	3	6	7	8		
迭代 10	1	1	3	3	3	3	6	7	8	8	
迭代 11	1	1	3	3	3	3	6	7	8	8	8
迭代 12	1	1	3	3	3	3	6	7	8	8	8	8

在后面，我们将基本算法*第* i 次迭代的第二项任务称为*长度平移（length-shift）*，即增加每个最大元素 x_j 满足 $x_i \le x_j < x_i\ c$ *的*子序列的长度。除了长度移位，改进后的算法与构造最长递增子序列的算法非常相似 [17]。如果我们使用数组来存储序列 (z_1，z_2，...), mim-

根据 [17] 中的算法，我们可以在 O（log k）时间内使用二进制搜索找到 x_i 的前代和 $x_i\ c$ 的前代。但是，长度平移（现在是通过使用 $z\ z_{j+1\ j}$ 依次覆盖指定范围内的每个 z_j ）需要 $O(k)$ 时间。这样一来，实现过程只需 $O(nk)$ 时间，仍未达到所宣称的约束。

示例

回到第 3 节的例子，c 为 2，输入序列为（7、15、2、14、14、6、8、11、17、15、14、13。我们在表 2 中显示了算法进行 12 次迭代后的两个数组：一个数组保存了 z_i 's （a 部分），另一个数组保存了 p_i 's 的索引（b 部分）。

该版本算法报告的 LaIS 将为（2、6、8、11、14、13）。

5.数据结构

为了达到所宣称的 $O(n \log k)$ 约束，内循环必须在 $O(\log k)$ 时间内执行。为了方便地实现长度平移，我们将 z_j 保存为一个有序链表，其中保存 z_j 的节点指向保存其后继元素 z_{j+1} 的节点。在每次迭代过程中，我们都会搜索 x_i 的前一个节点。然后在该位置后插入一个包含 x_i 的节点。要进行长度移动，需要确定相应的节点范围，并删除范围内最后一个节点的后继节点（如果存在）。为了构建子序列，我们仍然为 p_i 's 保留一个数组。该数组每次迭代修改一次，之后用于生成输出结果。

总之，算法的每次迭代都会执行：两次前代搜索、一次后代搜索、一次插入和一次可能的删除。

算法 1 LaIS 算法的伪代码。

```
1: for i ← 1 to n do
2:     v ← new_node()
3:     v.value ← x_i
4:     v.index ← i
5:     predecessor ← predecessor(x_i)
6:     if (pred ≠ null) then
7:         p_i ← pred.index
8:     其他
9:         P_i ← 0
10:    end if
11:    插入 (v)
12:    s ← successor(predecessor(x_i, c))
13:    if (s ≠ null) then
14:        删除
15:    end if
16: 结束 for
17: m ← tail_node().index
18: for i ← n down to m + 1 do
19:    if (x_m − c < x_i ≤ x_m) then
20:        打印 x_i
21:    end if
22: 结束 for
23: 打印 x_m
24: t ← m
25: while (p_t ≠ t) do
26:    for i = t − 1 down to p_t + 1 do
27:        if (x_{p_t} − c < x_i ≤ x_{p_t}) then
28:            打印 x_i
29:        end if
30:    结束
31:    打印 x_{p_t}
32:    t ← p_t
33: 结束 while
```

这些操作中的每一个都可以在 $O(\log k)$ 的平衡搜索结构时所需的时间

的节点。例如，我们可以使用 AVL 树[1]来实现每次操作*的* $O(\log k)$ 成本，或者使 用 splay 树[18]来实现每次操作的 $O(\log k)$ 成本。在实际应用中，最好保留一个指针链表，以便在恒定时间内找到给定节点的后继节点，在使用上述结构的情况下，每个节点至少需要三个指针。不过，从实用的角度来看，这种应用的最佳搜索结构是 jumplist [11]。使用 jumplist：前置搜索、插入和删除需要 $O(\log k)$ 的摊销时间（$O(\log k)$ 的预期时间 [7]），查找后继者需要恒定时间，而且我们只需使用和维护每个节点的两个指针。

致谢

我要感谢拉吉夫-拉曼（Rajiv Raman）和绍拉布-雷（Saurabh Ray）介绍了这个问题，并进行了多次有益的讨论。

参考资料

[1] G. Adelson-Velskii, E. Landis, On an information organization algorithm, Doklady Akademii Nauk SSSR 146 (1962) 263-266.

[2] M.Albert, R. Aldred, M. Atkinson, H. Ditmarsch, B. Handley, C. Handley, J. Opatrny, Longest Subences in permutations, Australian Journal of Combinatorics 28 (2003) 225-238.

[3] M.Albert, M. Atkinson, D. Nussbaum, J. Sack, N. Santoro, On the longest increasing subence of a circular list, Information Process- ing Letters 101 (2007) 55-59.

[4] M.Albert、A. Golnski、A. Hamel、A. Lopez-Ortiz、S. Rao、M. Sa- fari, Longest increasing subences in sliding windows, Theoretical Computer Science 321 (2004) 405-414.

[5] A.Apostolico, M. Atallah, S. Hambrusch, New clique and independent set algorithms for circle graphs, Discrete Applied Mathematics 36 (1992) 1-24.

[6] J.Baik、P. Deift、K. Johannsson，论随机排列最长递增子序列长度的分布，《美国数学学会学报》12 (1999) 1119-1178.

[7] H.Brönnimann, F. Cazals, M. Durand, Randomized jumplists：A jump-and-walk dictionary data structure, in：20th Symposium on Theoretical Aspects of Computer Science, in：LNCS, vol. 2607, 2003, pp.

[8] M.Chang, F. Wang, permu- tation graphs 上最大权重簇和最大权重独立集问题的有效算法，Information Processing Letters 76 (2000) 7-11.

[9] T.Cormen、C. Leiserson、R. Rivest、C. Stein，《算法导论》，第 2 版，麻省理工学院出版社，剑桥，2001 年。

[10] A.Delcher, S. Kasif, R. Fleischmann, J. Paterson, O. White, S. Salzberg, Alignment of whole genomes, Nucleic Acids Research 27 (1999) 2369-2376.

[11] A.Elmasry, Deterministic jumplists, Nordic Journal of Comput- ing 12 (1) (2005) 27-39.

[12] M.Fredman, On Computing the length of longest increasing subsequence, Discrete Mathematics 11 (1975) 29-35.

[13] J. Hunt, T. Szymanski, A fast algorithm for computing longest com-mon subences, Communications of the ACM 20 (1977) 350-353.

[14] D.Knuth, Permutations, Matrices, and generalized Young tableaux, Pacific Journal of Mathematics 34 (1970) 709-727.

[15] P.Ramanan, Tight $\Omega(n \log n)$ lower bound for finding a longest increasing subence, International Journal of Computer Mathemat- ics 65 (3-4) (1997) 161-164.

[16] G. Robinson, On representations of the symmetric group, American Journal of Mathematics 60 (1938) 745-760.

[17] C.Schensted, Longest increasing and decreasing Subsences, Cana- dian Journal of Mathematics 13 (1961) 179-191.

[18] D.Sleator, R. Tarjan, Self-adjusting binary search trees, Journal of the ACM 32 (3) (1985) 652-686.