

基于状态压缩的最长公共上升子序列快速算法

郭冬梅

(安徽理工大学 计算机科学与工程学院, 安徽 淮南 232001)

摘要:探讨了最长公共上升子序列(LCIS)问题,在前人算法的基础上提出一种高效求解 LCIS 的动态规划算法。对于 LCIS 问题,分别使用最长公共子序列(LCS)和最长上升子序列(LIS)相结合的算法、动态规划算法、经过状态压缩的改进动态规划算法进行设计,并对后两种算法进行了实现。设计的状态压缩的动态规划算法,实现了 LCIS 的快速求解。通过分析这三种算法的时间和空间复杂度,最终提出了时间复杂度为 $O(mn)$ 、空间复杂度为 $O(m)$ 或 $O(n)$ 的基于状态压缩的快速 LCIS 算法。

关键词:最长公共上升子序列;最长公共子序列;最长上升子序列;动态规划;状态压缩

中图分类号:TP391

文献标识码:A

文章编号:1673-629X(2014)05-0040-04

doi:10.3969/j.issn.1673-629X.2014.05.010

A Longest Common Increasing Subsequence Algorithm Based on State Compression

GUO Dong-mei

(College of Computer Science and Engineering, Anhui University of Science and
Technology, Huainan 232001, China)

Abstract: Discussed the problem of a Longest Common Increasing Subsequence (LCIS), put forward a fast dynamic algorithm for LCIS. For the LCIS problem, used respectively the algorithm of a Longest Common Subsequence (LCS) combined a Longest Increasing Subsequence (LIS) algorithm, dynamic programming algorithm, improved dynamic programming algorithm through state to design, and performed the second and the third algorithm. The designed state compressed dynamic programming algorithm realized a fast solution for LCIS. According to analyze the time and space complexity of the three algorithms, presented a fast algorithm for delivering a longest common increasing subsequence in $O(mn)$ time and $O(m)$ or $O(n)$ space finally.

Key words: LCIS; LCS; LIS; dynamic programming; state compression

0 引言

最长公共子序列(Longest Common Subsequence, LCS)问题和最长上升子序列(Longest Increasing Subsequence, LIS)问题在计算机科学算法领域中都是非常经典的问题,两者都是使用动态规划算法进行问题的求解。动态规划算法是求解决策过程的有效最优数学方法之一,为具有最优解结构性质的实际问题提供了行之有效的解决方法^[1-2]。利用最优解性质建立子问题最优值的递推关系可以自底向上递归地从子问题的最优解逐步构造出整个问题的最优解,大量降低计算所消耗的时间和空间复杂度^[3]。目前,将 LCS 和 LIS 两个问题融合在一起的最长公共上升子序列

(Longest Common Increasing Subsequence, LCIS)问题在国外已有了一定的研究基础,取得了相当的研究成果。通过前期收集资料、查阅文献等工作表明国内对 LCIS 问题的研究及应用处于起步阶段,鉴于此,文中在前人算法的基础上提出一种高效求解 LCIS 的快速动态规划算法。其问题描述:设 $A = \{a_1, a_2, \dots, a_m\}$ 和 $B = \{b_1, b_2, \dots, b_n\}$ 是长度分别为 m, n 的两个序列,这两个序列中的任意一对元素都是可以比较的。 A 和 B 的公共上升子序列满足条件如下: $\{a_{i_1} = b_{j_1}, a_{i_2} = b_{j_2}, \dots, a_{i_l} = b_{j_l}\}$, 其中 $i_1 < i_2 < \dots < i_l, j_1 < j_2 < \dots < j_l$, 同时对于所有的 $1 \leq k < l$, 都有 $a_{i_k} < a_{i_{k+1}}$ 。最长公共上升子序列是求 A 和 B 所有公共上升子序列中元素最多的那个子序列。

收稿日期:2013-07-10

修回日期:2013-10-17

网络出版时间:2014-02-11

基金项目:安徽省高等学校省级优秀青年人才基金项目(2011SQRL040);安徽理工大学青年教师科学研究基金(2012QNY31)

作者简介:郭冬梅(1982-),女,安徽怀远人,硕士,讲师,研究方向为计算机应用。

网络出版地址: <http://www.cnki.net/kcms/detail/61.1450.TP.20140211.1452.021.html>

最长上升子序列(LIS)、最长公共子序列(LCS)等组合数学中的问题已不是纯粹的理论问题。近年来,随着生物信息学的不断发展,LIS、LCS等问题已经在生物信息学领域得到了广泛的应用。例如:LCS问题已被应用到对两个DNA序列进行比较,从而求解基因之间的相似性^[4-6];根据LCS计算句子间的相似度模型、改进算法^[7-10]。LIS也已被应用到DNA序列的BLAST(Basic Local Alignment Search Tool)数据库和MUMmer系统中^[11],不仅如此,LCS也广泛应用在程序代码相似度、作业相似度等问题求解中。鉴于LIS、LCS的广泛应用,LCIS问题也将在生物信息学乃至其他领域崭露头角。该问题的提出将尝试应用于染色体序列等模式子序列问题的匹配和求解中,主要期望用于寻求三个乃至多个染色体序列中呈现上升趋势的最长MUMs子集^[12]。

鉴于LCIS算法在国内的文献资料中没有给出相关的算法分析与具体实现过程,考虑到其在生物学方面的重要应用前景,文中详细探索并设计构造了LCIS的几种不同算法,并分别对各自算法进行了详细描述。主要研究目标是要找出已知两个序列的最长公共子序列,且满足子序列的元素逐渐上升的条件。在对前人的算法进行研究的基础上,对之前采用的动态规划算法进行状态压缩,提出一种高效求解LCIS的快速算法,该算法将时间和空间复杂度降低到平方级以下。

1 最长公共上升子序列算法

求解最长公共上升子序列的关键就是要找出两个序列中长度最长的公共子序列,而且子序列中的元素是上升的。该问题求解涉及到LCS问题以及LIS问题,两者都属于经典的计算机算法与设计问题,均可使用动态规划算法。

对于LCIS问题,分别使用LCS和LIS相结合的算法、动态规划算法、经过状态压缩的改进动态规划算法进行设计,并对后两种算法进行了实现,通过分析这三种算法的时间和空间复杂度,提出了时间复杂度为 $O(mn)$ 、空间复杂度为 $O(m)$ 或 $O(n)$ 的快速LCIS算法。

1.1 转化为LCS和LIS相结合的算法

由于LCIS问题涉及到LCS和LIS算法,很容易想到使用LCS和LIS相结合的算法对其进行设计。对于集合 $A = \{a_1, a_2, \dots, a_m\}$ 中的每一项元素 a_i ,集合 $B = \{b_1, b_2, \dots, b_n\}$ 中任意元素 b_j ,当找到 $a_i = b_j$ 时,就搜索位于 a_i 和 b_j 元素之前的比 a_i 和 b_j 小的最长公共递增子序列。即对于 $A = \{a_1, a_2, \dots, a_{i-1}\}$ 和 $B = \{b_1, b_2, \dots, b_{j-1}\}$ 中的任意元素,找到其中相匹配的元素 a_{i_1} 和 b_{j_1} ,当 $a_{i_1} = b_{j_1}$ 时,满足 $a_{i_1} < a_i, b_{j_1} < b_j$,其中 $1 \leq i_1 < i$ 且

$$1 \leq j_1 < j。$$

设 $f[i][j]$ 表示到 $A = \{a_1, a_2, \dots, a_m\}$ 中前 i 个元素, $B = \{b_1, b_2, \dots, b_n\}$ 中前 j 个元素的LCIS长度。以下出现的 $f[i][j]$ 均按此定义表示。

(1) 若满足上述条件的元素存在,那么对所有的 i_1 和 j_1 ,目的就是要找出以 a_{i_1} 和 b_{j_1} 为末元素的所有公共上升子序列中的最大值即 $\max(f[i_1][j_1])$ 。那么, $f[i][j]$ 满足如下关系式:

$$f[i][j] = \max(f[i_1][j_1]) + 1 \quad (1)$$

上式表示使 $f[i_1][j_1]$ 最大的以 a_{i_1}, b_{j_1} 为末元素的两个公共递增子序列的最末端再加上 a_i 和 b_j ,即得到以 a_i 和 b_j 为末元素的最长公共上升子序列。

(2) 若不存在,只能满足 $a_i = b_j$ 这个条件,此时

$$f[i][j] = 1 \quad (2)$$

综合上面两种情况,在 $a_i = b_j$ 条件下,LCIS满足最优子结构,由此建立的递归关系式如下:

$$f[i][j] = \begin{cases} 0, i=0, j=0 \\ \max(f[i_1][j_1]) + 1, 1 \leq i_1 < i, 1 \leq j_1 < j, \\ \quad a_{i_1} = b_{j_1}, a_{i_1} < a_i \\ 1, 1 \leq i_1 < i, 1 \leq j_1 < j, a_{i_1} \neq b_{j_1} \end{cases} \quad (3)$$

(3) 由式(3)得到计算LCIS长度的算法1,描述如下:

输入:两个数组序列 $a[1 \cdots m], b[1 \cdots n]$;

输出: $f[i][j]$ 的值。

其中 i 和 j 分别表示数组序列 a 和 b 的下标,max表示最长公共上升子序列的长度。

```

1. for  $i = 1$  to  $m$  do
2. for  $j = 1$  to  $n$  do
3. if ( $a[i-1] = b[j-1]$ ) then
4. max := 0
5. for  $i_1 = 1$  to  $i$  do
6. for  $j_1 = 1$  to  $j$  do
7. if (( $a[i_1-1] = b[j_1-1]$ ) and ( $a[i_1-1] < a[i-1]$ ) and ( $f[i_1][j_1] > \max$ )) then
8. max :=  $f[i_1][j_1]$ 
9.  $f[i][j] := \max + 1$ 

```

从时间复杂度来看, i 每次从1循环到 m , j 均要从1执行到 n ,在满足条件的情况下, i_1 和 j_1 分别要再次从1执行到 i ,从1执行到 j 。因此该算法是一个四重循环,每个循环体内的计算量为 $O(1)$,由此总的复杂度为 $O(m^2n^2)$ 。

在空间复杂度上,由于需要对输入的两个序列进行比较,使用 $f[i][j]$ 存储LCIS的长度,占用的内存空间为 $O(mn)$,算法1的空间需求为 $O(mn)$ 。

由于上述算法时间复杂度达到了四次方,显然需

要对算法 1 进行改进。式(3)只考虑在 $a_i = b_j$ 条件下的最优子结构,把数组 a 和 b 看成两个对等序列,对数组 a 的每一项和数组 b 的每一项进行匹配。而对 $a_i \neq b_j$ 情况下递归关系式没有构建,从而导致算法耗时增加,数组的空间也没有得到充分利用。由于 LCIS 满足最优子结构性质,不论是在 $a_i = b_j$ 还是在 $a_i \neq b_j$ 条件下,均需构建相应的 LCIS 递归公式。

1.2 使用动态规划算法,构造求解 LCIS 递归公式

在 $a_i \neq b_j$ 条件下,不把 a 和 b 看成两个对等序列,考虑用数组 a 对数组 b 进行扫描,对数组 b 找最长上升子序列,从而把问题转化为在 $a[i-1] = b[j-1]$ 条件下,找数组 b 的 LIS。即在找数组 b 的 LIS 之前,两个元素 $a[i-1]$ 和 $b[j-1]$ 已经是匹配的。 a_i 与 b_j 不相等的情况包含了 $a[i-1] = b[j-1]$ 因素,可将 $a[i-1] = b[j-1]$ 纳入到 $a_i \neq b_j$ 进行统一考虑,这充分利用了 LCIS 具有最优子结构的性质。由于对 b 进行扫描,在 $a_i \neq b_j$ 条件下, $f[i][j]$ 的初始值为 $f[i-1][j]$,即把上一层($i-1$)层的结果作为当前层的初始值。在当前层即 i 层 $a_i = b_j$ 相等来临之前,让 $a[i-1]$ 之前的所有元素与 $b[1 \cdots k]$ ($1 \leq k \leq j$) 进行匹配,找出使公共上升子序列长度最长的 j 值,将其赋值给 k 。此时, $f[i][j] = f[i][k] + 1$ 。那么,在 $a_i = b_j$ 相等来临时,根据 $f[i-1][j]$ 和 $f[i][j]$ 的比较结果,取其中较大的值作为 $f[i][j]$ 的最终结果。

综合考虑 $a_i = b_j$ 以及 $a_i \neq b_j$ 两种情况,根据前述分析,构建递归公式也即状态转移方程如下:

$$f[i][j] = \max\{f[i-1][j], \max\{f[i][k] + 1 \mid 1 \leq k \leq j, b[k] < b[j]\}\} \quad (4)$$

由式(4)可得对算法 1 进行改进的算法 2,描述如下:

输入: m 个数组成的集合序列 $A = \{a_1, a_2, \dots, a_m\}$; n 个数组成的集合序列 $B = \{b_1, b_2, \dots, b_n\}$;

输出: $f[i][j]$ 的值。

```

1. for  $i = 1$  to  $m$  do
2. for  $j = 1$  to  $n$  do
3.  $f[i][j] := f[i-1][j]$ 
4.  $k := 0$ 
5. if  $((a[i-1] > b[j-1]) \text{ and } (f[i][k] > f[i][j]))$  then
6.  $k := j$ 
7. if  $((a[i-1] = b[j-1]) \text{ and } (f[i][j] < f[i][k] + 1))$  then
8.  $f[i][j] := f[i][k] + 1$ 

```

从时间复杂度来看, i 每次从 1 循环到 m , j 均要从 1 执行到 n , 由于每个循环体内的计算量为 $O(1)$, 由此算法耗时为 $O(mn)$ 。经过改进的算法 2 的时间复杂度

降低到平方级。

在空间复杂度上,仍需要二维空间存储 LCIS 的长度,算法 2 的空间需求为 $O(mn)$ 。

1.3 使用状态压缩,对算法 2 进行改进

算法 2 是数组 a 在 i 处固定的情况下对数组 b 找寻 LIS,可进一步转化为在数组 a 中寻找满足 LIS 的元素,将式(4)中数组 b 的下标 j 省去,此时需满足数组 a 的元素个数 m 大于或等于 b 的元素个数 n ;反之亦然。使用一维数组存储两数组的 LCIS 长度和子序列,从而使状态从二维空间压缩至一维空间。

定义:一维数组 f 存放最长公共上升子序列的长度。 $f_temp[i]$ 标记前一个与自身相同的数在数组 $a[i]$ 中的位置。数组 $LCIS[k]$ 用于存放公共上升子序列。

其算法设计如下:

(1) $a[i] \neq b[j]$ 时, $f[i]$ 记录在相等来临前最长公共上升子序列的长度。

(2) $a[i] = b[j]$ 时,在相等来临时判断是取 $f[i-1] + 1$ 的值还是取 $f[f_temp[i]]$ 的值。其中又讨论两种情况:若 $f_temp[i]$ 存在且 $f[f_temp[i]]$ 大于等于 $f[i-1]$ 即 $f[i-1] \leq f[f_temp[i]]$, 令 $f[i] = f[f_temp[i]]$;若 $f_temp[i]$ 不存在,直接令 $f[i] = f[i-1] + 1$ 。

(3) 令 $f[k] = \max, k = m$, 若 $f[k] = \max$ 并且小于栈顶元素,则 $a[k]$ 入栈, $\max = \max - 1, k = k - 1$, 直至 $\max = 0$ 为止,取出栈中元素组成一个 LCIS 序列。

考虑 $m \geq n$ 的情况,构造递归公式如下:

$$f[i] = \begin{cases} f[f_temp[i]], f_temp[i] \text{ 存在} \\ \text{且 } f[i-1] \leq f[f_temp[i]] \\ f[i-1] + 1, f_temp[i] \text{ 不存在} \end{cases} \quad (5)$$

由式(5)得到算法 3,描述如下:

输入:两个序列: $A = \{a_1, a_2, \dots, a_m\}$ 、 $B = \{b_1, b_2, \dots, b_n\}$, 且满足 $m \geq n$ 。

// 以下是求解 $f[i]$ 的值

```

1. for  $j = 1$  to  $n$  do
2.  $\max := 0$ 
3. for  $i = 1$  to  $m$  do
4. if  $(a[i] < b[j])$  then
5. if  $(\max < f[i])$  then
6.  $\max := f[i]$ 
7. if  $(a[i] = b[j])$  then
8. if  $((f\_temp[i]! = -1) \text{ and } (\max \leq f[f\_temp[i]]))$  then //  $f\_temp[i]$  存在情况下与  $\max$  比较
9.  $f[i] := f[f\_temp[i]]$ 
10. else //  $f\_temp[i]$  不存在情况下,将  $\max + 1$  赋值给  $f[i]$ 

```

```
11.  $f[i] := \max + 1$ 
// 以下是输出数组  $a$  和  $b$  的最长公共上升子序列
长度  $\max$  值
12.  $\max := 0$  // 初始化
13. for  $i = 1$  to  $m$  do
14. if ( $\max < f[i]$ ) then
15.  $\max := f[i]$ 
16. print  $\max$ 
// 以下是输出 LCIS 序列
17. TYPE  $s = \text{RECORD}$  // 定义栈结构
    elem: ARRAY[1..1 000] of elemtp;
    top: 0..1 000
18.  $k := m$ 
19. while ( $\max < > 0$ ) do
20. if (( $f[k] = \max$ ) and(  $a[k] < (s.\text{elem}[s.$ 
top])))) then //  $f[k] = \max$  且  $a[k]$  小于栈顶元素
21. push_stack( $s, a[k]$ ) // 将  $a[k]$  入栈
22.  $k := k - 1, \max := \max - 1$ 
23. for  $i = 1$  to  $m$  do
24. LCIS[ $i$ ] := pop_stack( $s$ ) // 当  $\max = 0$  时, 栈
中所有元素出栈, 并依次存放在数组 LCIS 中
```

由于算法3将两个数组 a 和 b 合二为一考虑, 将问题转化为从数组 a 中找出和数组 b 元素相同且满足上升条件的子集, 使用一维数组存储长度及 LCIS, 从而实现空间状态压缩一倍, 空间复杂度降低至 $O(m) (m \geq n)$ 。

算法3的时间复杂度与算法2相比, 并没有改变, 仍是 $O(mn)$ 。

2 三种算法的时间与空间复杂度分析

LCIS 的算法1和算法2相比, 考虑到 LCIS 属于经典的动态规划问题, 算法2充分利用了动态规划算法具有的最优子结构性性质, 将算法1的时间复杂度从 $O(m^2n^2)$ 降至 $O(mn)$ 。而算法3又在此基础上, 使用状态压缩对算法2存储空间进行了压缩, 构造状态转移方程, 将空间复杂度从 $O(mn)$ 降至 $O(m) (m \geq n)$ 或 $O(n) (n \geq m)$ 。

(上接第39页)

Research challenges[C]//Proc of the 1st European workshop on software architecture. Andrews:Spring-Verlag, 2004:200-205.

[9] 黄 罡, 王千祥, 梅 宏, 等. 基于软件体系结构的反射式中间件研究[J]. 软件学报, 2003, 14(11):1819-1826.

[10] 朱 庆, 王小平, 薛小平, 等. 基于构件的网构软件系统动态演化[J]. 计算机工程, 2010, 36(1):55-57.

3 结束语

文中探讨了求解最长公共上升子序列(LCIS)算法问题。对于 LCIS 问题, 通过对上述三种算法的时间和空间复杂度进行比较, 最终提出使用经过状态压缩的改进动态规划算法进行分析设计, 可将求解 LCIS 问题的算法时间和空间复杂度控制在二次方范围内, 从而有效降低了求解此类问题的时间和空间复杂度, 为后续的应用推广奠定了基础。

参考文献:

[1] Alsuwaiyel M H. Algorithms design techniques and analysis [M]. Beijing: Publishing House of Electronics Industry, 2003.

[2] Cooper R. Dynamic programming: An overview[EB/OL]. 2001. <http://econ. bu. edu/faculty/cooper/Dynprog/introlect. pdf>.

[3] 王晓东. 计算机算法设计与分析[M]. 第2版. 北京: 电子工业出版社, 2005.

[4] 王映龙, 杨炳儒, 宋泽锋, 等. 基因序列相似程度的 LCS 算法研究[J]. 计算机工程与应用, 2007, 43(31):45-47.

[5] 张晓敏, 陈 昊, 明 仲. 寻找序列的变化内容[J]. 计算机工程与应用, 2011, 47(31):49-52.

[6] 胡 婕, 业 宁, 罗晓波, 等. 多序列的近似 LCS 改进算法[J]. 计算机工程, 2011, 37(2):166-168.

[7] 冷强奎, 秦玉平, 王春立. 基于句子相似度的论文抄袭检测模型研究[J]. 计算机工程与应用, 2011, 47(24):199-201.

[8] 金 博, 史彦军, 腾弘飞. 基于篇章结构相似度的复制检测算法[J]. 大连理工大学学报, 2007, 47(1):125-130.

[9] 林贤明, 李堂秋, 陈毅东. 句子相似度的动态规划求解及改进[J]. 计算机工程与应用, 2004, 40(35):64-65.

[10] 王 品, 黄广君. 信息检索中的句子相似度计算[J]. 计算机工程, 2011, 37(12):38-40.

[11] 严华平, 李 刚, 张建宏. 生物信息挖掘中 LIS 算法研究[J]. 计算机应用研究, 2009, 6(1):62-63.

[12] Yang I-Hsuan, Huang Chien-Pin, Chao Kun-Mao. A fast algorithm for computing a longest common increasing subsequence[J]. Information Processing Letters, 2005, 93:249-253.

[11] 吴 波. 面向构件的软件系统动态配置技术的研究[D]. 北京: 北京工业大学, 2010.

[12] 余 萍, 马晓星, 吕 建, 等. 一种面向动态软件体系结构的在线演化方法[J]. 软件学报, 2006, 17(6):1360-1371.

[13] 李玉龙, 李长云. 软件动态演化技术[J]. 计算机技术与发展, 2008, 18(9):83-86.

[14] 管贤春. 基于构件化软件的动态演化研究与应用[D]. 广州: 广东工业大学, 2010.

基于状态压缩的最长公共上升子序列快速算法

作者:

郭冬梅, [GUO Dong-mei](#)

作者单位:

[安徽理工大学 计算机科学与工程学院, 安徽 淮南, 232001](#)

刊名:

[计算机技术与发展](#) 

英文刊名:

[Computer Technology and Development](#)

年, 卷(期):

2014(5)

本文链接: http://d.g.wanfangdata.com.cn/Periodical_wjz201405010.aspx