Available online at www.sciencedirect.com
ScienceDirect

Journal of Complexity 24 (2008) 173 – 184

Journal of COMPLEXITY

www.elsevier.com/locate/jco

Finding a longest common subsequence between a run-length-encoded string and an uncompressed string☆

J.J. Liu[a], Y.L. Wang[b,*], R.C.T. Lee[b]

[a] *Department of Information Management, National Taiwan University of Science and Technology, Taipei, Taiwan, ROC*
[b] *Department of Computer Science and Information Engineering, National Chi Nan University, Nantou, 1 University Rd. Puli, Nantou Hsien, Taiwan 545, ROC*

Received 5 August 2006; accepted 14 June 2007
Available online 24 July 2007

Abstract

In this paper, we propose an $O(\min\{mN, Mn\})$ time algorithm for finding a longest common subsequence of strings X and Y with lengths M and N, respectively, and run-length-encoded lengths m and n, respectively. We propose a new recursive formula for finding a longest common subsequence of Y and X which is in the run-length-encoded format. That is, $Y = y_1 y_2 \cdots y_N$ and $X = r_1^{l_1} r_2^{l_2} \cdots r_m^{l_m}$, where r_i is the repeated character of run i and l_i is the number of its repetitions. There are three cases in the proposed recursive formula in which two cases are for r_i matching y_j. The third case is for r_i mismatching y_j. We will look specifically at the prior two cases that r_i matches y_j. To determine which case will be used when r_i matches y_j, we have to find a specific value which can be obtained by using another of our proposed recursive formulas.

Keywords: Longest common subsequence; Run-length encoding; String compression

1. Introduction

Let X and Y be two strings over a finite alphabet set Σ, where $X = x_1x_2x_3 \ldots x_M$ and $Y = y_1y_2y_3 \ldots y_N$. Substrings $x_1x_2x_3 \ldots x_i$ and $y_1y_2y_3 \ldots y_j$ are represented by X_i and Y_j, respectively, where $1 \leqslant i \leqslant M$ and $1 \leqslant j \leqslant N$. A *subsequence* of a string is obtained by deleting zero or

☆ This work was supported in part by the National Science Council of the Republic of China under Contract NSC 95-2221-E260-025.

* Corresponding author.
E-mail address: yuelwang@ncnu.edu.tw (Y.L. Wang).

0885-064X/$ - see front matter © 2007 Elsevier Inc. All rights reserved.
doi:10.1016/j.jco.2007.06.003

some (not necessarily consecutive) characters of this string. A *common subsequence* of X and Y is a subsequence in both X and Y. A *longest common subsequence* of X and Y is a common subsequence with the maximum length. Let $LCS(i, j)$ denote the length of a longest common subsequence of X_i and Y_j. $LCS(i, j)$ can be computed by the following recursive formula:

$$LCS(i,j) = \begin{cases} LCS(i-1, j-1) + 1 & \text{if } x_i = y_j, \\ \max\{LCS(i-1, j), LCS(i, j-1)\} & \text{if } x_i \neq y_j, \end{cases}$$

with initial conditions $LCS(i, 0) = LCS(0, j) = LCS(0, 0) = 0$, for $1 \leqslant i \leqslant M$ and $1 \leqslant j \leqslant N$.

For the sake of convenience, we call the above formula *the standard formula* of an LCS algorithm. The table which contains all of the values of $LCS(i, j)$, for $i = 1, 2, \ldots, M$, $j = 1, 2, \ldots, N$, is called *an LCS table*. An LCS table has a property in which the difference between any two consecutive values in an LCS table must be between -1 and 1. Moreover, the value in an element with a larger row/column index is greater than or equal to the value in a position with a smaller row/column index. We call the above property *the ascending property* of an LCS table.

The time-complexity of an LCS algorithm which uses the standard formula to solve the longest common subsequence problem is proportional to the production of the lengths of two uncompressed strings, namely $O(MN)$. Advanced algorithms have been proposed for finding LCS of two strings with time complexity: $O(LN + N \log N)$ and $O(L(M + 1 - L) \log N)$ [5] (where L is the length of an LCS), $O((r + N) \log N)$ [6] (where r is the total number of matching pairs of X and Y), $O(N \log |\Sigma| + d \log \log \min\{d, MN/d\})$ [3] (where d is the number of so-called dominant matches, and $|\Sigma|$ is the cardinality of the alphabet set), or $O((M+N)|\Sigma| + |M| \log |M|)$ [2] (where $|M|$ is the cardinality of the set of so-called maximal matches between substrings of X and Y). Note that in the worst case both d and r are of size $\Omega(N^2)$ and L is always bounded by N.

There is a string compression technique which is called *run-length encoding* [7]. In a string, the maximal repeated string of characters is called a *run* and the number of repetitions is called the *run-length*. Thus, a string can be encoded more compactly by replacing a run by a single instance of the repeated character along with its run-length. Compressing a string in this way is called *run-length encoding* and a run-length encoding string is abbreviated as an *RLE* string. For example, the RLE string of string *bdcccaaaaaa* is $b^1d^1c^3a^6$. Note that, in run-length encoding, X denotes $r_1^{l_1} r_2^{l_2} \ldots r_m^{l_m}$, where r_j is the repeated character of run j and l_j is its corresponding run-length, for $j = 1, 2, \ldots, m$. The position of the last character of run r_j in the corresponding uncompressed string is denoted by $lp(r_j)$ which is equal to $l_1 + l_2 + \cdots + l_j$.

Freschi and Bogliolo [4] proposed an $O(mN + Mn - mn)$ time algorithm for finding the longest common subsequence of two *RLE* strings, where M and N are the lengths of the original strings X and Y, respectively, and m and n are the numbers of runs in the *RLE* representations of X and Y, respectively. Apostolico et al. [1] gave another algorithm for solving the same problem in $O(mn \log(mn))$ time. In this paper, we shall propose an $O(\min\{mN, Mn\})$ time algorithm for finding the longest common subsequence between an *RLE* string and an uncompressed string. In the remainder of this paper, we shall assume, without loss of generality, that X is an *RLE* string and Y is an uncompressed string and $O(\min\{mN, Mn\})$ is equal to $O(mN)$.

This paper is organized as follows. In Section 2, we describe some important properties of an LCS table. In Section 3, we present our algorithm for solving the LCS problem under the condition that one of the two strings, say X, is in the *RLE* form. Conclusions and open problems are given in Section 4.

2. Some properties of an LCS table

Let $\tau \in \Sigma$ be a symbol in Y. The position of the ith τ in Y is denoted by $Y_\tau(i)$. The inverse function of Y_τ is denoted by Y_τ^{-1}. The value of Y_τ^{-1} is called the *rank* of τ. Thus, if $Y_\tau(i) = j$, then $Y_\tau^{-1}(j) = i$, i.e., the rank of the τ in y_j is i. The position just before $Y_\tau(i)$ in Y, i.e., $Y_\tau(i) - 1$, is denoted by $pre(Y_\tau(i))$. Moreover, $pre^k(Y_\tau(j))$ denotes $pre(Y_\tau(j-k+1))$ for $1 \leqslant k \leqslant j$ while $pre^0(Y_\tau(j)) = Y_\tau(j)$. For instance, $Y = y_1 y_2 \cdots y_8 = baaaccaa$. Then, $Y_a(1) = 2$, $Y_a^{-1}(2) = 1$, $Y_a(2) = 3$, $Y_a^{-1}(3) = 2$, $Y_a(3) = 4$, $Y_a^{-1}(4) = 3$, $Y_a(4) = 7$, $Y_a^{-1}(7) = 4$, $Y_a(5) = 8$, and $Y_a^{-1}(8) = 5$. Furthermore, $pre^0(Y_a(4)) = 7$, $pre^1(Y_a(4)) = 6$, $pre^2(Y_a(4)) = pre(Y_a(4-2+1)) = pre(Y_a(3)) = 3$, and so on.

For $X = r_1^{l_1} r_2^{l_2} \cdots r_m^{l_m}$, we are interested especially in the computations of the rows with index $lp(r_i)$, $i = 1, 2, \ldots, m$, of an LCS table, if we are only given the values in row $lp(r_{i-1})$. In the following, we shall introduce some lemmas which will be used in computing the values of an LCS table for these specific rows. The notations r_i, l_i, and y_j used in the following lemmas denote the character of the ith run of string X, the length of run i, and the jth character of string Y, respectively.

Lemma 1. *For run i and y_j, if r_i mismatches y_j, then $LCS(lp(r_i), j) = \max\{LCS(lp(r_i), j-1), LCS(lp(r_{i-1}), j)\}$.*

Proof. If r_i mismatches y_j, then all characters that are in run r_i will mismatch y_j. This implies that $LCS(lp(r_i), j)$ is obtained by either ignoring the jth symbol of Y or the entire run r_i of X. Therefore, $LCS(lp(r_i), j) = \max\{LCS(lp(r_i), j-1), LCS(lp(r_{i-1}), j)\}$. □

Lemma 2. *For run i and y_j, if r_i matches y_j, then*

$$LCS(lp(r_i), j) = \max_{0 \leqslant k \leqslant l_i} \{LCS(lp(r_{i-1}), pre^k(j)) + k\}.$$

Proof. To compute $LCS(lp(r_i), j)$ directly from row $lp(r_{i-1})$ when r_i matches y_j, we can consider all possible numbers of the characters in run i used to match the same number of the same character in Y. If no character in run i is used to match the same character in Y in order to get $LCS(lp(r_i), j)$, then $LCS(lp(r_i), j) = LCS(lp(r_{i-1}), j) = LCS(lp(r_{i-1}), pre^0(j))$. If there are k characters, $1 \leqslant k \leqslant l_i$, in run i which are used to match the same number of the same characters in Y to obtain $LCS(lp(r_i), j)$, then $LCS(lp(r_i), j) = LCS(lp(r_{i-1}), pre^k(j)) + k$. However, we do not know which $LCS(lp(r_{i-1}), pre^k(j)) + k$, $0 \leqslant k \leqslant l_i$, has the largest value before we compare all of them. Therefore, $LCS(lp(r_i), j) = \max_{0 \leqslant k \leqslant l_i}\{LCS(lp(r_{i-1}), pre^k(j)) + k\}$. □

See Fig. 1 for an elucidation of Lemma 2. To compute $LCS(4, 4)$ directly from row 1, all the possible values needed for consideration are $LCS(1, 4) + 0$, $LCS(1, 3) + 1$, $LCS(1, 2) + 2$, and $LCS(1, 0) + 3$. The maximum value is 3 (= $LCS(1, 2) + 2$ or $LCS(1, 0) + 3$).

We call element $(lp(r_{i-1}), pre^k(j))$ an *originating element* and k an originating value of element $(lp(r_i), j)$ in the LCS table if $LCS(lp(r_{i-1}), pre^k(j)) + k = LCS(lp(r_i), j)$, for $k = 0, 1, \ldots, l_i$. Sometimes there are more than one originating element of element $(lp(r_i), j)$. For example, in Fig. 1, both elements $(lp(r_1), pre^3(4))$ and $(lp(r_1), pre^2(4))$ are originating elements of element $(lp(r_2), 4)$. An originating element with the smallest k among all originating elements with respect to element $(lp(r_i), j)$ in an LCS table is called the *nearest originating element* of element $(lp(r_i), j)$ and k is called the *nearest originating value* of element $(lp(r_i), j)$. With the concept

		∈	a	b	a	a
	∈	0	0	0	0	0
$lp\ (r_1) \longrightarrow$	b	0	0	1	1	1
	a	0	1	1	2	2
	a	0	1	1	2	3
$lp\ (r_2) \longrightarrow$	a	0	1	1	2	3

Fig. 1. An LCS table for $X = baaa$ and $Y = abaa$.

of originating elements and Lemmas 1 and 2, the recursive formula for computing the longest common subsequence can be represented as follows:

$$LCS(lp(r_i), j) = \begin{cases} \max\{LCS(lp(r_i), j-1), LCS(lp(r_{i-1}), j)\} & \text{if } r_i \neq y_j, \\ LCS(lp(r_{i-1}), pre^k(j)) + k & \text{if } r_i = y_j, \end{cases}$$

where element $(lp(r_{i-1}), pre^k(j))$ is an originating element of element $(lp(r_i), j)$ in the LCS table. Note that the above recursive formula only computes the values in the rows with $lp(r_i)$, $i = 1, 2, \ldots, m$.

In the above formula, the most time consuming step occurs at computing the LCS value of two matched characters. It will take $O(l_i)$ time for finding an originating element of element $(lp(r_i), j)$ so that the total time complexity becomes $O((l_1 + l_2 + \cdots + l_m)N) = O(MN)$. In the following, we shall introduce some properties in a standard LCS table. By using these properties, we can compute the longest common subsequence between a run-length-encoded string and an uncompressed string efficiently.

For run i and y_j of a standard LCS table, an element (t, j) is called a *critical element* of element $(lp(r_i), j)$ if $LCS(t, j) = LCS(lp(r_i), j)$, where $lp(r_{i-1}) \leqslant t \leqslant lp(r_i) - 1$. The value of a critical element, i.e. $LCS(t, j)$, is called a *critical value* with respect to element $(lp(r_i), j)$. Let $S_{i,j}$ denote the number of all critical elements of element $(lp(r_i), j)$. We use the LCS table in Fig. 1 to illustrate the above notation. The RLE string X is $b^1a^3 = r_1^{l_1}r_2^{l_2}$. Thus, $lp(r_2) = 1 + 3 = 4$. It means that the position of the last character of a^3 in the original string X is 4. In column 4 of the LCS table, we can find that there is only one critical element with respect to element (4, 4), namely element (3, 4), and the critical value is $LCS(3, 4) = 3$. Therefore, $S_{2,4} = 1$. We can also find that $S_{2,0} = 3$, $S_{2,1} = 2$, $S_{2,2} = 3$, and $S_{2,3} = 2$. Lemmas 3 and 4 will describe the relation between $LCS(lp(r_i), j)$ and $S_{i,j}$.

Lemma 3. *For run i and y_j, if r_i matches y_j and $S_{i,j-1} = 0$, then $LCS(lp(r_i), j) = LCS(lp(r_i), j-1)$.*

Proof. $S_{i,j-1} = 0$ means that there is no critical element with respect to element $(lp(r_i), j-1)$ of an LCS table. Thus, $LCS(lp(r_i) - 1, j-1) = LCS(lp(r_i), j-1) - 1$. According to the standard LCS formula and the above equality, if r_i matches y_j, then

$$\begin{aligned} LCS(lp(r_i), j) &= LCS(lp(r_i) - 1, j-1) + 1 \\ &= LCS(lp(r_i), j-1) - 1 + 1 \\ &= LCS(lp(r_i), j-1). \qquad \square \end{aligned}$$

		∈	a	b	a	a	a	a
	∈	0	0	0	0	0	0	0
$lp\,(r_1)$ ⟶	b	0	0	1	1	1	1	1
	a	0	1	1	2	2	2	2
	a	0	1	1	2	3	3	3
$lp\,(r_2)$ ⟶	a	0	1	1	2	3	4	4

Fig. 2. An LCS table for $X = baaa$ and $Y = abaaaa$.

See Fig. 2 for an illustration of Lemma 3. There is no critical element with respect to element (4, 5) of the LCS table in Fig. 2. Thus, $S_{2,5} = 0$ and $LCS(3, 5) = LCS(4, 5) - 1$. Since r_2 matches y_6, $LCS(4, 6) = LCS(3, 5) + 1 = LCS(4, 5) - 1 + 1 = LCS(4, 5)$.

Lemma 4. *For run i* and y_j, *if r_i matches y_j and $S_{i,j-1} > 0$, then $LCS(lp(r_i), j) = LCS(lp(r_i), j - 1) + 1$.*

Proof. $S_{i,j-1} > 0$ implies that there exists at least one critical element with respect to run i and y_{j-1}. Thus, $LCS(lp(r_i), j - 1) = LCS(lp(r_i) - 1, j - 1)$. According to the above equation and the standard LCS formula again, if r_i matches y_j, then

$$\begin{aligned} LCS(lp(r_i), j) &= LCS(lp(r_i) - 1, j - 1) + 1 \\ &= LCS(lp(r_i), j - 1) + 1. \qquad \square \end{aligned}$$

See element (4, 4) of the LCS table in Fig. 2 for an illustration. We can find that $S_{2,3} = 2$. Therefore, $LCS(3, 3) = LCS(4, 3) = 2$. Since r_2 matches y_4, $LCS(4, 4) = LCS(3, 3) + 1 = LCS(4, 3) + 1 = 2 + 1 = 3$.

Lemmas 5–7 describe a way for computing $S_{i,j}$, where Lemma 5 can be obtained directly from the definition of $S_{i,j}$.

Lemma 5. *For run i and y_j, if $LCS(lp(r_i), j) = LCS(lp(r_{i-1}), j)$, then $S_{i,j} = l_i$.*

Lemma 6. *For run i and y_j, if r_i mismatches y_j and $LCS(lp(r_i), j) > LCS(lp(r_{i-1}), j)$, then $S_{i,j} = S_{i,j-1} < l_i$.*

Proof. At first, we prove that if r_i mismatches y_j and $LCS(lp(r_i), j) > LCS(lp(r_{i-1}), j)$, then $S_{i,j-1} < l_i$. For the purpose of contradiction, we assume that $S_{i,j-1} = l_i$. It means that there are l_i critical elements with respect to run i and y_{j-1} and $LCS(lp(r_{i-1}), j - 1) = LCS(lp(r_i), j - 1)$. By the ascending property of an LCS table, $LCS(lp(r_{i-1}), j) \geqslant LCS(lp(r_{i-1}), j - 1)$. Thus, $LCS(lp(r_{i-1}), j) \geqslant LCS(lp(r_i), j - 1)$. However, by Lemma 1, if r_i mismatches y_j, then $LCS(lp(r_i), j) = \max\{LCS(lp(r_i), j - 1), LCS(lp(r_{i-1}), j)\} = LCS(lp(r_{i-1}), j)$. It contradicts the given constraint $LCS(lp(r_i), j) > LCS(lp(r_{i-1}), j)$. Therefore, $S_{i,j-1} < l_i$.

Now, we are at a position to prove that if r_i mismatches y_j and $LCS(lp(r_i), j) > LCS(lp(r_{i-1}), j)$, then $S_{i,j} = S_{i,j-1}$. Assume that $S_{i,j-1} = k < l_i$. Then, by a similar reasoning as the proof of Lemma 1, we can obtain $LCS(lp(r_i), j) = LCS(lp(r_i) - 1, j) = \cdots = LCS(lp(r_i) - k, j) = LCS(lp(r_i), j - 1)$ and $LCS(lp(r_i) - k - 1, j) = \max\{LCS(lp(r_i) - k - 1, j - 1),$

		∈	b	a	a	a	b	a	a	a
	∈	0	0	0	0	0	0	0	0	0
	b	0	1	1	1	1	1	1	1	1
$lp\,(r_1) \longrightarrow$	b	0	1	1	1	1	2	2	2	2
	a	0	1	2	2	2	2	3	3	3
	a	0	1	2	3	3	3	3	4	4
$lp\,(r_2) \longrightarrow$	a	0	1	2	3	4	4	4	4	5

Fig. 3. An LCS table for $X = bbaaa$ and $Y = baaabaaa$. (a) an rp table; (b) its corresponding reverse rp table.

$LCS(lp(r_{i-1}), j)\}$. It is clear that $LCS(lp(r_i) - k - 1, j - 1)$ is less than $LCS(lp(r_i), j) - 1$. By the following derivation, we can obtain $LCS(lp(r_i) - k - 1, j) < LCS(lp(r_i), j)$:

$$\begin{aligned} LCS(lp(r_i) - k - 1, j) &= \max\{LCS(lp(r_i) - k - 1, j - 1), LCS(lp(r_{i-1}), j)\} \\ &= \max\{LCS(lp(r_i), j) - 1, LCS(lp(r_{i-1}), j)\} \\ &= LCS(lp(r_i), j) - 1 \\ &< LCS(lp(r_i), j). \end{aligned}$$

This completes the proof. □

An illumination of Lemma 6 can be seen from Fig. 3. Since r_2 mismatches y_5 and $LCS(5, 5) > LCS(2, 5)$, $S_{2,5} = S_{2,4} = 0$.

Lemma 7. *For run i and y_j, if r_i matches y_j, then $S_{i,j} = l_i - k$, where k is the nearest originating value of element (i, j) in the LCS table.*

Proof. Since k is the nearest originating value of element $(lp(r_i), j)$, by definition, $LCS(lp(r_i), j) = LCS(lp(r_{i-1}), pre^k(j)) + k$ and $LCS(lp(r_{i-1}) + k, j) \geqslant LCS(lp(r_{i-1}), pre^k(j)) + k$. However, by the ascending property of an LCS table, $LCS(lp(r_{i-1}) + k, j) \leqslant LCS(lp(r_i), j)$. This implies that $LCS(lp(r_{i-1}) + k, j) = LCS(lp(r_i), j)$.

If we can prove that $LCS(lp(r_{i-1}) + k - 1, j) < LCS(lp(r_{i-1}) + k, j)$, then the lemma follows. Viewing the first $k - 1$ characters of run i as a new run, the nearest originating value, say h, of element $(lp(r_{i-1}) + k - 1, j)$ must be less than k. By the definition of an originating value,

$$\begin{aligned} LCS(lp(r_{i-1}) + k - 1, j) &= LCS(lp(r_{i-1}), pre^h(j)) + h \\ &< LCS(lp(r_{i-1}), pre^k(j)) + k \\ &= LCS(lp(r_i), j). \end{aligned}$$

Therefore, there are exactly $l_i - k$ critical elements with respect to element $(lp(r_i), j)$ and $S_{i,j} = l_i - k$. □

See Fig. 3 for an example of Lemma 7. Since r_2 matches y_7 and $LCS(lp(r_2), 7) = \max_{0 \leqslant h \leqslant 3} \{LCS(lp(r_1), pre^h(7)) + h\} = LCS(lp(r_1), pre^2(7)) + 2 = LCS(lp(r_1), pre^3(7)) + 3 = 4$, $k = 2$ is the nearest originating value of element $(lp(r_2), 7)$. Thus, $S_{2,7} = l_2 - k = 3 - 2 = 1$.

	∈	b	a	a	a	b	a	a	a
∈	0	0	0	0	0	0	0	0	0
$lp\,(r_1) \longrightarrow b^2$	0	1	1	1	1	2	2	2	2
$lp\,(r_2) \longrightarrow a^3$	0	1	2	3	4	4	4	4	5
Providing values			1	0	-1		-1	-2	-3
Clue values			1	1	1		0	-1	-1

Fig. 4. An illustrating example.

From Lemmas 3 to 7, computing $LCS(lp(r_i), j)$ and $S_{i,j}$ only uses the values in row $lp(r_{i-1})$ of an LCS table for $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, N$. We rename the table which contains only the rows $lp(r_1), lp(r_2), \ldots, lp(r_m)$ of an LCS table as an *RULCS table*. Note that row $lp(r_i)$ of an LCS table is the ith row of an RULCS table. Thus, many definitions based on an LCS table can also be applied to an RULCS table. For example, if element $(lp(r_{i-1}), q)$ is an originating element of element $(lp(r_i), j)$ in an LCS table, then we still call element $(i-1, q)$ an originating element of element (i, j) in an RULCS table. For instance, the corresponding RULCS table of Fig. 3 is shown in Fig. 4 without including the last two rows.

We summarize the above lemmas as the following theorem.

Theorem 1. *For run i and y_j,*

$$RULCS(i,j) = \begin{cases} \max\{RULCS(i,j-1), RULCS(i-1,j)\} & \text{if } r_i \neq y_j, \\ RULCS(i,j-1) & \text{if } r_i = y_j \text{ and } S_{i,j-1} = 0, \\ RULCS(i,j-1)+1 & \text{if } r_i = y_j \text{ and } S_{i,j-1} > 0, \end{cases}$$

and

$$S_{i,j} = \begin{cases} l_i & \text{if } RULCS(i,j) = RULCS(i-1,j), \\ S_{i,j-1} & \text{if } r_i \neq y_j \text{ and } RULCS(i,j) > RULCS(i-1,j), \\ l_i - k & \text{if } r_i = y_j, \end{cases}$$

where k is the nearest originating value of the element (i, j) in the RULCS table. The initial conditions are $S_{i,0} = l_i$ and $RULCS(i,0) = RULCS(0,j) = RULCS(0,0) = 0$, for $1 \leqslant i \leqslant m$ and $1 \leqslant j \leqslant N$.

Clearly, in Theorem 1, the computations of $RULCS(i, j)$ and $S_{i,j}$ can be done intuitively in constant time for all conditions except the last condition for computing $S_{i,j}$. In the next section, we shall show that the last condition for computing $S_{i,j}$ can also be obtained in constant time.

3. An efficient algorithm for finding the nearest originating value

If we try to find the nearest originating value of an element by an exhaustive search, then the time-complexity of our approach will go back to $O(MN)$. The property described in Lemma 8 can be used to find the nearest originating element of an element in constant time. By applying the property in Lemma 8, we can use the inverted list technique so that the nearest originating element can be found in constant time.

Lemma 8. *For run i and y_j in an RULCS table, let $r_i = \tau$, $Y_\tau^{-1}(j) = p$, and an integer q where $p - l_i + 1 \leqslant q \leqslant p$. Then the following conditions are equivalent*:

(1) *Element $(i-1, pre(Y_\tau(q)))$ is an originating element of element (i, j).*
(2) *$p - q + 1$ is an originating value of element (i, j).*
(3) *$RULCS(i, j) - p = RULCS(i-1, pre(Y_\tau(q))) + 1 - q$.*

Proof. (1)⇒(2) Since element $(i-1, pre(Y_\tau(q)))$ and element $(i-1, pre^{p-q+1}(Y_\tau(p)))$ are the same element, by the definition of an originating value, $p - q + 1$ is an originating value of element (i, j).

(2)⇒(3) If $p-q+1$ is an originating value of element (i, j), then, by definition, $RULCS(i, j) = RULCS(i-1, pre^{p-q+1}(Y_\tau(p))) + p - q + 1$. Rearranging the equality, we can obtain $RULCS(i, j) - p = RULCS(i-1, pre(Y_\tau(q))) + 1 - q$.

(3)⇒(1) After replacing $pre(Y_\tau(q))$ by $pre^{p-q+1}(Y_\tau(p))$ and then rearranging the formula in condition (3), we have $RULCS(i, j) = RULCS(i-1, pre^{p-q+1}(Y_\tau(p))) + (p - q + 1)$.

According to the definition of an originating element, we know that $(i-1, pre^{p-q+1}(Y_\tau(p)))$ is an originating element of element (i, j). Therefore, element $(i-1, pre(Y_\tau(q)))$ is also an originating element of element (i, j) and the Lemma follows. □

An illumination of Lemma 8 can be seen from Fig. 4. Since $RULCS(2, 7) = RULCS(1, pre^2(7)) + 2 = RULCS(1, pre(Y_a(4))) + 2$, element $(1, pre(Y_a(4)))$ is an originating element of element $(2, 7)$ and $q = 4$. Since $p = Y_a^{-1}(7) = 5$ and $q = 4$, $p - q + 1 = 2$ is an originating value of element $(2, 7)$. Moreover, $RULCS(2, 7) - p = 4 - 5 = -1$ is equal to $RULCS(1, pre(Y_a(4))) + 1 - q = 2 + 1 - 4 = -1$.

For run i, we call $RULCS(i, j) - p$ and $RULCS(i-1, pre(Y_\tau(p))) + 1 - p$ the *clue value* and the *providing value*, respectively, of the pth τ in Y, where $Y_\tau(p) = j$. For example, see run 2 and y_6 in Fig. 4. Since y_6 is the fourth a in Y, $RULCS(i-1, pre(Y_a(p))) + 1 - p = RULCS(2-1, pre(Y_a(4))) + 1 - 4 = RULCS(1, pre(6)) - 3 = RULCS(1, 5) - 3 = 2 - 3 = -1$. Therefore, the providing value of the fourth a for run 2 is -1. We can find that the providing value of the third a for run 2 is also -1. The other providing values can be seen from the last row of Fig. 4. The clue value of the fifth a for run 2 is $RULCS(i, j) - p = RULCS(2, 7) - 5 = 4 - 5 = -1$. By Lemma 8, we can obtain the information that both elements $(1, 3)$ and $(1, 5)$ are originating elements of element $(2, 7)$ and their corresponding originating values are 3 and 2, respectively.

Corollary 9. *For run i in an RULCS table, let $r_i = \tau$, α be the providing value of the qth τ, β be the clue value of the pth τ, and $p - l_i + 1 \leqslant q \leqslant p$. If element $(i-1, pre(Y_\tau(q)))$ is an originating element of element $(i, Y_\tau(p))$, then $\alpha \pmod{l_i} \equiv \beta \pmod{l_i}$.*

We can view the providing values as a table whose indexes are the ranks of τ in Y and the corresponding value of each index is its providing value. See Fig. 5(a) for an example in which the providing values in Fig. 4 are represented by a table. We call the above table an *rp table* of run i. In order to find the nearest originating value of element (i, j) in constant time, we can construct an *inverted rp table* for run i. That is, the indexes of an inverted rp table are providing values and the corresponding value of each index is the rank of τ. Note that if more than one τ have the same index, i.e., the same providing value, then a larger rank will replace a smaller rank in an inverted rp table. For example, the inverted rp table of Fig. 5(a) is shown in Fig. 5(b). We can see from the a

The rank of a	1	2	3	4	5	6
Providing value	1	0	-1	-1	-2	-3

b

Providing value	-3	-2	-1	0	1
The rank of a	6	5	4	2	1

Fig. 5. The rp table and reverse rp table of Fig. 3.

table in Fig. 5(a) that both the third and the fourth a have the same providing value, namely -1, and the final stored value is 4 in Fig. 5(b).

To illuminate that the nearest originating value of element (i, j) can be found in constant time (see Function `NearestOriginatingValue`), we shall compute the nearest originating value of element (2, 7) as an example by using the inverted rp table in Fig. 5(b). We know that the clue value of element (2, 7) is $\mathit{RULCS}(2,7) - Y_\tau^{-1}(7) = 4 - 5 = -1$. Using -1 as the index of the inverted rp table in Fig. 5(b), we can obtain 4, namely the fourth a, from the table. Then, by Lemma 8, the nearest originating value of element (2, 7) is $p - q + 1 = 5 - 4 + 1 = 2$.

By using the modulus operation, the indexes of an inverted rp table can be bounded in the range from 0 to $l_i - 1$. That is what Lemma 10 states.

Function `NearestOriginatingValue`(*int i, int j*)

```
begin
    $theClueValue = RULCS(i, j) - Y_\tau^{-1}(j)$;
    return $Y_\tau^{-1}(j) - rp[theClueValue] + 1$;
end
```

Lemma 10. *For run i, let $r_i = \tau$, ω be the providing value of the sth τ, and let β be the clue value of the pth τ, and let element $(i-1, pre(Y_\tau(q)))$ be the nearest originating element of element $(i, Y_\tau(p))$. If $q < s \leqslant p$, then $\omega \pmod{l_i} \not\equiv \beta \pmod{l_i}$.*

Proof. By definition, the providing value of the sth τ is $\mathit{RULCS}(i-1, pre(Y_\tau(s))) + 1 - s$. We know that $s > q$. Then

$$\mathit{RULCS}(i-1, pre(Y_\tau(s))) \geqslant \mathit{RULCS}(i-1, pre(Y_\tau(q))). \tag{1}$$

Since element $(i-1, pre(Y_\tau(q)))$ is the nearest originating element of $(i, Y_\tau(p))$, we also have

$$\mathit{RULCS}(i, Y_\tau(p)) = \mathit{RULCS}(i-1, pre(Y_\tau(q))) + p - q + 1 \tag{2}$$

and

$$\mathit{RULCS}(i, Y_\tau(p)) > \mathit{RULCS}(i-1, pre(Y_\tau(s))) + p - s + 1 \quad \text{for } q < s \leqslant p. \tag{3}$$

From (2) and (3), we can obtain

$$\mathit{RULCS}(i-1, pre(Y_\tau(q))) - q + 1 > \mathit{RULCS}(i-1, pre(Y_\tau(s))) - s + 1. \tag{4}$$

From (1) and (4), we can derive

$$\begin{aligned} &\mathit{RULCS}(i-1, pre(Y_\tau(q))) - q + 1 - (\mathit{RULCS}(i-1, pre(Y_\tau(s))) - s + 1) \\ &\quad \leqslant \mathit{RULCS}(i-1, pre(Y_\tau(s))) - q + 1 - (\mathit{RULCS}(i-1, pre(Y_\tau(s))) - s + 1) \\ &\quad = s - q < p - q \leqslant l_i. \end{aligned} \tag{5}$$

By exchanging l_i and $RULCS(i-1, pre(Y_\tau(s))) - s + 1$ in (5), we can have the following derivation:

$$RULCS(i-1, pre(Y_\tau(q))) - q + 1 - l_i < RULCS(i-1, pre(Y_\tau(s))) - s + 1. \tag{6}$$

Using (4) and (6), we can find that $RULCS(i-1, pre(Y_\tau(q))) - q + 1 - l_i < RULCS(i-1, pre(Y_\tau(s))) - s + 1 < RULCS(i-1, pre(Y_\tau(q))) - q + 1$. This implies that $RULCS(i-1, pre(Y_\tau(q))) - q + 1 \pmod{l_i} \not\equiv RULCS(i-1, pre(Y_\tau(s))) - s + 1 \pmod{l_i}$. Since $RULCS(i-1, pre(Y_\tau(q))) - q + 1 = RULCS(i, pre(Y_\tau(p))) - p = \beta$, and $RULCS(i-1, pre(Y_\tau(s))) - s + 1$ is the providing value of the sth τ, i.e. ω, $\beta \pmod{l_i} \not\equiv \omega \pmod{l_i}$. □

See Fig. 4 for an illumination of Lemma 10. For run 2, the clue value of the 6th a is $RULCS(2, Y_a(6)) - 5 = RULCS(2, 8) - 6 = -1$. Element $(1, pre(Y_a(4)))$ is the nearest originating element of element $(2, Y_a(6))$ because $RULCS(1, pre(Y_a(4))) + 1 - 4 = -1 = RULCS(2, Y_a(6)) - 5$. The providing value of the 5th a is $RULCS(1, pre(Y_a(5))) + 1 - 5 = -2$. It is clear that $-2 \pmod{l_2} \not\equiv -1 (\bmod\, l_2)$.

For simplicity, we still call an inverted rp table with modulus an inverted rp table if it will not make confusion. We use the computation of $S_{2,7}$ as an example to show how to use the inverted rp table. The table in Fig. 6 is the inverted rp table after computing $RULCS(2, 7)$. The clue value of the fifth a in run 2 is -1, and 2 after the modulus operation. Thus, by retrieving the inverted rp table, the providing value of the fourth a in Y is also 2. Then, by Theorem 1, $S_{2,7} = l_2 - k = 3 - (5 - 4 + 1) = 1$.

Algorithm A is used for finding a longest common subsequence of a run-length-encoded string and an uncompressed string.

Algorithm A.

```
Input: An RLE string $X = r_1^{l_1} r_2^{l_2} \cdots r_m^{l_m}$ and an uncompressed string $Y = y_1 y_2 \cdots y_N$.
Output: A longest common subsequence $\mathcal{L}$.
begin
    Step 1. Use the recursive formula in Theorem 1 to compute $S_{i,j}$, for $i = 1, 2, \ldots, m$
        and $j = 1, 2, \ldots, N$.
    Step 2. Let $\mathcal{L} = \phi$, $i = m$, and $j = N$.
    Step 3. If $i = 0$ or $j = 0$, then output $\mathcal{L}$ in the reverse order and terminate.
    Step 4. For $r_i$ and $y_j$, adjust $i$, $j$, and $\mathcal{L}$ according to the following cases.
        Case 1. $r_i$ matches $y_j$.
            Append $l_i - S_{i,j}$ characters $r_i$ to $\mathcal{L}$ and change $i$ and $j$ to $i - 1$ and
            $pre^{l_i - S_{i,j}}(j)$, respectively.
        Case 2. $r_i$ mismatches $y_j$ and $S_{i,j} = l_i$.
            Replace $i$ by $i - 1$.
        Case 3. $r_i$ mismatches $y_j$ and $S_{i,j} < l_i$.
            Replace $j$ by $j - 1$.
    Step 5. Go to Step 3.
end
```

We use an example to illustrate Algorithm A. See Fig. 7 for computing $S_{i,j}$ of $X = b^2a^3$ and $Y = baaabaaa$. After Step 1, the resulting $S_{i,j}$, $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, N$ are shown in Fig. 7. And, the changes of i, j, and $\mathcal{L}$ in each iteration are shown in Table 1. We summarize the above results as the following theorem.

The providing value with modulus operation	0	1	2
The rank of *a*	2	5	4

Fig. 6. The inverted rp table for run 2 after computing *RULCS*(2, 7).

	∈	b	a	a	a	b	a	a	a
∈	0	0	0	0	0	0	0	0	0
b^2	2	1	1	1	1	0	0	0	0
a^3	3	3	2	1	0	0	0	1	0

Fig. 7. The $S_{i,j}$ of the RULCS table.

Table 1
The changes of i, j, and $\mathcal{L}$

i	j	$\mathcal{L}$
2	8	ϕ
1	5	*aaa*
0	0	*aabb*

Theorem 2. *A longest common subsequence of an RLE string X and an uncompressed string Y can be found in $O(\min\{mN, nM\})$ time by Algorithm A.*

Proof. The way in Algorithm A to obtain a longest common subsequence is similar to that of obtaining a longest common subsequence from a standard LCS table. Clearly, it only takes $O(m + N)$ time to obtain a longest common subsequence if an RULCS table is given. According to Theorem 1 and Lemma 8, an RULCS table can be built in $O(mN)$ time. This completes the proof. □

4. Concluding remarks

In this paper, we proposed an $O(\min\{mN, nM\})$ time algorithm for solving the longest common subsequence problem of a run-length-encoded string and an uncompressed string. The following table shows the time complexities for solving the LCS problem by using Freschi's algorithm [4], Apostolico's algorithm [1], and our proposed algorithm in different compressed conditions. In Table 2, *T1* denotes the time complexity if both strings X and Y can be compressed efficiently, *T2* denotes the time complexity if only one of X and Y can be compressed efficiently, and *T3* denotes the time complexity if the lengths of run-length-encoded strings X and Y are nearly equal to the lengths of the original strings X and Y, respectively.

We can see from Table 2 that our algorithm is the most efficient algorithm when only one string can be compressed efficiently. It is obvious that our algorithm can be parallelized in a PRAM EREW computational model by using $O(\min\{m, N\})$ processors and takes $O(m + N)$ time. A general version of this problem is to consider both strings which are in *RLE* form. We are trying to solve this problem now.

Table 2
The comparisons of different algorithms for the LCS problem with run-length-encoded strings.

	Freschi's algorithm [4]	Apostolico's algorithm [1]	Our algorithm
T1	$O(mN + Mn - mn)$	$O(mn \log(mn))$	$O(\min\{mN, Mn\})$
T2	$O(MN)$	$O(mN \log(mN))$	$O(mN)$
T3	$O(MN)$	$O(MN \log(MN))$	$O(MN)$

References

[1] A. Apostolico, G.M. Landau, S. Skiena, Matching for run-length encoded strings, Journal of Complexity 15 (1) (1999) 4–16.

[2] B.S. Baker, R. Giancarlo, Sparse dynamic programming for longest common subsequence from fragments, J. Algorithms 42 (2002) 231–254.

[3] D. Eppstein, Z. Galil, R. Giancarlo, G.F. Italiano, Sparse dynamic programming I: linear cost functions, J. ACM 39 (3) (1992) 519–545.

[4] V. Freschi, A. Bogliolo, Longest common subsequence between run-length-encoded strings: a new algorithm with improved parallelism, Inform. Process. Lett. 90 (4) (2004) 167–173.

[5] D.S. Hirschberg, Algorithms for the longest common subsequence problem, J. ACM 24 (4) (1977) 664–675.

[6] J.W. Hunt, T.G. Szymanski, A fast algorithm for computing longest common subsequences, Commun. ACM 20 (5) (1977) 350–353.

[7] K. Sayoood, E. Fow (Eds.), Introduction to Data Compression, second ed., Morgan Kaufmann, Los Altos, CA, 2000.