Theoretical Computer Science 964 (2023) 113944

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Linear-space S-table algorithms for the longest common subsequence problem [☆], [☆☆], [☆☆☆]

Bi-Shiang Lin [a], Kuo-Si Huang [b], Chang-Biau Yang [a,*]

[a] *Department of Computer Science and Engineering, National Sun Yat-sen University, Kaohsiung, Taiwan*
[b] *Department of Business Computing, National Kaohsiung University of Science and Technology, Kaohsiung, Taiwan*

ARTICLE INFO

Article history:
Received 12 July 2022
Received in revised form 21 April 2023
Accepted 9 May 2023
Available online 12 May 2023
Communicated by T. Calamoneri

Keywords:
Longest common subsequence
S-table
Linear space
Concatenation and merging
Incremental update

ABSTRACT

Given two sequences A and B of lengths m and n, respectively, the *consecutive suffix alignment* (CSA) problem is to compute the *longest common subsequence* (LCS) between A and each suffix of B. The data structure of two-dimensional matrix for solving the CSA problem is named S-table. The S-table would be infeasible due to its quadratic-space requirement for long sequences. The linear-space S-table, proposed by Alves et al. (2005), consists of the first row of the S-table and the changes between every two consecutive rows. However, there is no further discussions and practical applications for the linear-space S-table. This paper proposes algorithms for improving three problems, related to LCS and S-table, by using the linear-space S-table. Suppose that $A = A^{(1)}A^{(2)}$ (concatenation of two substrings), and we are given the S-table of $A^{(2)}$ and B, as well as the alignment result (LCS length) of $A^{(1)}$ to each prefix of B. The *concatenated LCS* (CoLCS) problem is to find the alignment result of A and B. For the CoLCS problem, this paper proposes an $O(n)$-time algorithm with the technique of set union-find to achieve the linear space. Next, for merging two linear S-tables, we propose an $O(n \log n)$-time algorithm with the technique of range query and update to improve the quadratic-time algorithm using quadratic-space S-tables. Then, we propose an $O(n)$-time algorithm to update the linear-space S-table of A and B to obtain the linear-space S-table of $A\sigma$ and B, where σ denotes a new character appended to the tail of A. For further applications to long sequences by using the linear-space S-table, the algorithms proposed in this paper are essential and necessary.

1. Introduction

The *longest common subsequence* (LCS) problem [2–11] is a fundamental method for estimating the similarity between sequences. The LCS problem has been extensively studied for several decades since 1970. It can be solved in $O(mn)$ time [7] by the *dynamic programming* (DP) approach, where m and n denote the lengths of the two input sequences, respectively. Lots of variant LCS problems were also proposed, such as the *merged longest common subsequence* problem [12–16], which

[☆] This article belongs to Section A: Algorithms, automata, complexity and games, Edited by Paul Spirakis.

[☆☆] This research work was partially supported by the Ministry of Science and Technology of Taiwan under contract MOST 104-2221-E-110-018-MY3.

[☆☆☆] A preliminary version of this paper was presented at the 35th Workshop on Combinatorial Mathematics and Computation Theory, Taichung, Taiwan, 2018 [1].

[*] Corresponding author.
E-mail address: cbyang@cse.nsysu.edu.tw (C.-B. Yang).

https://doi.org/10.1016/j.tcs.2023.113944
0304-3975/© 2023 Elsevier B.V. All rights reserved.

considers the LCS with a merged sequence, and the *constrained LCS* problem [17–22], which computes the LCS with a constrained sequence. Furthermore, some parameters, such as the alphabet size and the similarity of input sequences, can be used to analyze tighter bounds of the LCS problem [23–25].

The *consecutive suffix alignment* (CSA) problem is one of the variant LCS problems. Given two sequences A and B, the CSA problem is to compute the LCS length between A and each suffix of B [26], where a suffix means a substring starting at a certain position and ending at the last position of a string. The *S-table* can be used to solve the problem in quadratic space. The CSA problem can be used in various applications, such as the common substring alignment problem [27–29], the cyclic string comparison between two strings or between A and each suffix of B [30–32]. In 2003, Landau et al. [29] proposed a linear time algorithm with the given S-table to solve the common substring alignment problem.

Landau et al. [26] proposed two algorithms to solve the CSA problem. One solves the problem in $O(nL)$ time and space for a fixed alphabet size, and the other solves the problem in $O(nL + n\log|\Sigma|)$ time and $O(n)$ space, where $|\Sigma|$ and L denote the alphabet size and the LCS length, respectively. Alves et al. [33] proposed another algorithm in $O(mn)$ time and $O(n)$ space for the CSA problem. In addition, Alves et al. [33] proposed the linear-space S-table, which consists of the first row of the S-table and the changes between every two consecutive rows. However, there is no further discussions and practical applications for the linear-space S-table.

In this paper, we propose four algorithms for solving three LCS-related problems (only the LCS length is computed) by using the linear-space S-table as follows. Let $A = A^{(1)}A^{(2)}$ (concatenation of two substrings). Suppose that we already have the S-table of $A^{(2)}$ and B, as well as the alignment result of $A^{(1)}$ to each prefix of B. The *concatenated LCS* (CoLCS) problem is to calculate the alignment result of A and B. The alignment result corresponds to the first row in the S-table, reflecting the LCS length of A to each prefix of B. It can be applied to further concatenations.

(1) We first propose one algorithm for solving the CoLCS problem in $O(n\log n)$ time, where n denotes the length of input sequences. This algorithm demonstrates our concept for solving the CoLCS problem.

(2) We improve the CoLCS algorithm with the range maximum query, implemented by the set union-find operation. Then, the time complexity is reduced to $O(n)$. The above two algorithms have been presented in a local workshop [1].

(3) We propose an $O(n\log n)$-time algorithm to merge two linear-space S-tables, instead of merging two S-tables with a straightforward method in $O(nL)$ time.

(4) Finally, we propose an algorithm in $O(n)$ time to compute the linear-space S-table of $A\sigma$ and B when given the linear-space S-table of A and B, where σ denotes a new character appended to the tail of A.

The organization of this paper is given as follows. Section 2 introduces the preliminary knowledge of the LCS problem, CSA problem, CoLCS problem and S-table. Next, the above four algorithms are presented in Sections 3 through 6, respectively. Finally, the conclusions are given in Section 7.

2. Preliminaries

2.1. The grid directed acyclic graph for the longest common subsequence problem

A sequence of characters is denoted by an upper-case letter, such as A or B. Suppose a sequence $A = a_1a_2\cdots a_m$. Then, $|A| = m$ denotes the length of A, $A_{i..j}$ denotes the substring of A from index i to j, and $A_{i..j} = \emptyset$ if $i > j$. A subsequence of A is obtained by deleting an arbitrary number of characters (not necessarily consecutive) in A.

Definition 1 *(LCS).* Given two sequences $A = a_1a_2\cdots a_m$ and $B = b_1b_2\cdots b_n$ with lengths m and n, respectively, the *longest common subsequence* (LCS) problem is to find a common subsequence between A and B with the maximal length.

For example, the LCS of $A = \texttt{cggattctgt}$ and $B = \texttt{tctgatggt}$, denoted as $LCS(A, B)$, is $\texttt{cgatgt}$ with length 6. The LCS problem can be solved by the grid directed acyclic graph (GDAG) [28] in $m+1$ rows and $n+1$ columns, where rows and columns are labeled from 0 to m and 0 to n, respectively. Each vertex in the GDAG is denoted as $G(i, j)$, where $0 \le i \le m$ and $0 \le j \le n$. The LCS problem can be transformed to compute a path with the largest total weight from $G(0, 0)$ to $G(m, n)$. Fig. 1 shows an example. In the GDAG, the weight of each diagonal edge from $G(i-1, j-1)$ to $G(i, j)$ is 1 if $a_i = b_j$, and 0 if $a_i \ne b_j$, the weight of edge from $G(i, j-1)$ to $G(i, j)$, or from $G(i-1, j)$ to $G(i, j)$, is 0, where $1 \le i \le m$ and $1 \le j \le n$.

Definition 2 *($P_G(i, j)$).* For $0 \le i \le m$ and $0 \le j \le n$, $P_G(i, j)$ is the value of the largest weight path from $G(0, 0)$ to $G(i, j)$.

With the GDAG, the LCS length of $A_{1..i}$ and $B_{1..j}$ is equal to $P_G(i, j)$. The DP method for solving the LCS problem [11] can now be rewritten as Equation (1), whose time complexity is still $O(mn)$.

$$P_G(i, j) = \max \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0, \\ P_G(i-1, j-1) + 1 & \text{if } a_i = b_j, \\ \max \begin{cases} P_G(i-1, j) \\ P_G(i, j-1) \end{cases} & \text{if } a_i \ne b_j. \end{cases} \tag{1}$$

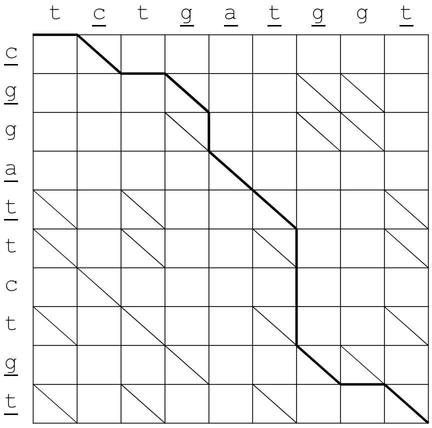


Fig. 1. The grid directed acyclic graph (GDAG) for solving the LCS problem with $A = \texttt{cggattctgt}$ and $B = \texttt{tctgatggt}$. Here, the path formed with thick lines is the LCS solution (`cgatgt`). Note that the diagonal edges with weight 0 are not shown.

Table 1
The matrix W_G of $A = \texttt{ttct}$ and $B = \texttt{tctgatggt}$.

i \ j	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	3	3	3	3	3	3
1	0	0	1	2	2	2	2	2	2	3
2	0	0	0	1	1	1	2	2	2	3
3	0	0	0	0	0	0	1	1	1	2
4	0	0	0	0	0	0	1	1	1	2
5	0	0	0	0	0	0	1	1	1	2
6	0	0	0	0	0	0	0	0	0	1
7	0	0	0	0	0	0	0	0	0	1
8	0	0	0	0	0	0	0	0	0	1
9	0	0	0	0	0	0	0	0	0	0

2.2. The consecutive suffix alignment problem and the S-table

Definition 3 *(CSA [26]).* Given two sequences A and B, the *consecutive suffix alignment* (CSA) problem is to compute the alignment of A and each suffix of B (each $B_{i..n}$, $0 \le i \le n$).

With the DP formula in Equation (1) for the LCS problem, $|LCS(A, B_{1..j})|$ can be computed in $O(mn)$ time for all $1 \le j \le n$. It is not so efficient to compute each $|LCS(A, B_{i..j})|$, for $0 \le i \le j \le n$ using the DP approach repeatedly. In the GDAG, the CSA problem can be transformed to finding the maximal weight path from $G(0, i)$ to $G(m, j)$ for $0 \le i \le j \le n$.

Definition 4 *($W_G(i, j)$ [33]).* For $0 \le i \le j \le n$, $W_G(i, j)$ is the maximal weight of all the paths from $G(0, i)$ to $G(m, j)$.

With Definition 4, it is clear that $W_G(i, j) = |LCS(A, B_{i+1..j})|$. Table 1 shows an example of W_G with $A = \texttt{ttct}$ and $B = \texttt{tctgatggt}$.

By inspecting W_G, some properties can be summarized as follows.

- For each row in W_G, the value starts from 0.
- For each row in W_G, the values from left to right are nondecreasing.
- For each row in W_G, the difference between two consecutive values is either 0 or 1.

With the above three properties, W_G can be represented by Table 2, denoted as S-table S.

Definition 5 *(S-table [29]).* For $0 \le i \le n$, $S_{i,0} = i$. For $0 \le i \le n$ and $0 < k \le L$, where L is the maximal value in W_G, $S_{i,k}$ is the minimum of j for which $W_G(i, j) = k$. If no such j exists, $S_{i,k} = \infty$.

The first element in $S_{i,*}$ (row i of S) is $S_{i,0} = i$, and each remaining element in $S_{i,*}$ records the index of the column which is the leftmost of each LCS length appearing in row i of W_G. For example, in row $i = 3$ of W_G, the leftmost $k = 1$ appears at the column $j = 6$, so $S_{3,1} = 6$. Alves et al. proposed and proved the following property of S-table [33].

Table 2
The S-table S of $A = \texttt{ttct}$ and $B = \texttt{tctgatggt}$, where the starting index i indicates $B_{i+1..n}$, and each element d_i in column D means that it is the first occurrence in row i.

		Length k				
	S	0	1	2	3	D
Starting index i	0	0	1	2	3	
	1	1	2	3	$\underline{9}$	$d_1 = 9$
	2	2	3	$\underline{6}$	9	$d_2 = 6$
	3	3	6	9	∞	$d_3 = \infty$
	4	$\underline{4}$	6	9	∞	$d_4 = 4$
	5	$\underline{5}$	6	9	∞	$d_5 = 5$
	6	6	9	∞	∞	$d_6 = \infty$
	7	$\underline{7}$	9	∞	∞	$d_7 = 7$
	8	$\underline{8}$	9	∞	∞	$d_8 = 8$
	9	9	∞	∞	∞	$d_9 = \infty$

Theorem 1 *([33]). For $0 \le i \le n-1$ in S,*

1. *Exactly one element of $S_{i,*}$ does not appear in $S_{i+1,*}$, which is $S_{i,0} = i$.*
2. *At most one element with a finite value in $S_{i+1,*}$ does not appear in $S_{i,*}$.*

With Theorem 1, the S-table can be represented as row 0 of S-table and the new members appearing in the latter rows of every two consecutive rows.

Definition 6 *(Linear-space S-table [33]).* Let $D = d_1 d_2 \cdots d_n$. For $1 \le i \le n$, d_i records the element which appears in $S_{i,*}$ but not in $S_{i-1,*}$. If there is no such new element, we set $d_i = \infty$. The *linear-space S-table* means the row zero of the S-table ($S_{0,*}$) and D.

An example of D is shown in the rightmost column of Table 2. The 2-D S-table can be constructed from $S_{0,*}$ and D. Therefore, the solution of the CSA problem can be represented with the linear-space S-table [33].

2.3. Solving the concatenated LCS problem with the 2-D S-table

We use an example to explain how to solve the LCS problem of multiple common substrings with the S-table [28,29]. Suppose $A = \texttt{cggattctgt}$ and $B = \texttt{tctgatggt}$, where A is formed by concatenating three substrings $A^{(1)} = \texttt{cgga}$, $A^{(2)} = \texttt{ttct}$ and $A^{(3)} = \texttt{gt}$. In more general situations, $A^{(r)}$ for $1 \le r \le 3$ may appear several times to form a longer sequence A. Note the S-table of $A^{(2)}$ and B has been already established in Table 2. Fig. 2 shows the GDAG of A and B, which is composed of three subgraphs, corresponding to $A^{(1)}$, $A^{(2)}$ and $A^{(3)}$, respectively. The alignment result I of the first subgraph can be viewed as the input of the second subgraph, and the alignment result O of the second subgraph can be viewed as the input of the third subgraph.

Let $G^{(r)}$ denote subgraph r, whose input and output are denoted as $I^{(r)}$ and $O^{(r)}$, respectively. In addition, let $S^{(r)}$ denote the S-table of $A^{(r)}$ and B. The goal is to get the output $O^{(r)}$ with the input $I^{(r)}$ and S-table $S^{(r)}$. The following DP formula can be easily obtained [28,29].

$$O_j^{(r)} = \max_{0 \le i \le j} \{I_i^{(r)} + W_{G^{(r)}}(i, j)\}. \tag{2}$$

For example, $O_6^{(2)} = 4 = \max\{0+3, 0+2, 1+2, 1+1, 2+1, 3+1, 3+0\}$. As one can see, only the leftmost position of $I^{(r)}$ with value k is needed to compute $O^{(r)}$. Let position PI_k denote the smallest index in $I^{(r)}$ with value k, and position $PO_{k'}$ denote the smallest index in $O^{(r)}$ with value k'. In this example, $PI = \langle 0, 2, 4, 5 \rangle$ and $PO = \langle 0, 1, 2, 3, 6, 9 \rangle$. Now, $O^{(r)}$ can be represented as PO in Equation (3) [29].

$$PO_{k'} = \min\{j | k + W_{G^{(r)}}(PI_k, j) = k', 0 \le k \le k' \text{ and } 0 \le j \le n\}. \tag{3}$$

With the S-table $S^{(r)}$, Equation (3) can be transformed into Equation (4).

$$PO_{k'} = \min_{0 \le k \le k'} \{S_{PI_k, k'-k}^{(r)}\}, \text{ if } S_{PI_k, k'-k}^{(r)} \text{ exists.} \tag{4}$$

For example, the smallest column index of value 4 in $O^{(2)}$, denoted by PO_4, is obtained by $\min\{S_{PI_0,4}^{(2)}, S_{PI_1,3}^{(2)}, S_{PI_2,2}^{(2)}, S_{PI_3,1}^{(2)}, S_{PI_4,0}^{(2)}\} = \min\{S_{0,4}^{(2)}, S_{2,3}^{(2)}, S_{4,2}^{(2)}, S_{5,1}^{(2)}, S_{\infty,0}^{(2)}\} = \min\{\varnothing, 9, 9, 6, \varnothing\} = 6$, where symbol $\varnothing$ denotes that it does not

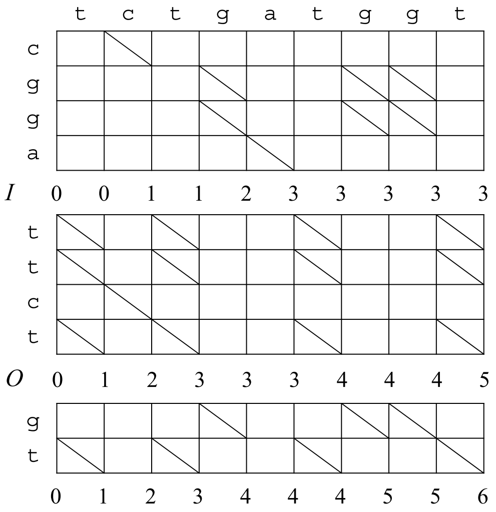


Fig. 2. The GDAG composed of three subgraphs with $A^{(1)} = \texttt{cgga}$, $A^{(2)} = \texttt{ttct}$, $A^{(3)} = \texttt{gt}$ and $B = \texttt{tctgatggt}$.

Table 3
An example of matrix M for finding column minima by using S-table in Table 2, where the diagonal $M_{k,k}$ and the bottom row correspond to $PI = \langle 0, 2, 4, 5 \rangle$ and $PO = \langle 0, 1, 2, 3, 6, 9 \rangle$ in Fig. 2, respectively.

M		Length k': 0	1	2	3	4	5
Row index k	0	0	1	2	3		
	1		2	3	6	9	
	2			4	6	9	
	3				5	6	9
Minimum		0	1	2	3	6	9

exist in Table 2. It means that $|LCS(A^{(1)}, B_{1..2})| + |LCS(A^{(2)}, B_{3..9})| = 1 + 3 = 4$, $|LCS(A^{(1)}, B_{1..4})| + |LCS(A^{(2)}, B_{5..9})| = 2 + 2 = 4$, and $|LCS(A^{(1)}, B_{1..5})| + |LCS(A^{(2)}, B_{6..6})| = 3 + 1 = 4$. And 6 is the leftmost index of B to get LCS length 4.

Definition 7 *(M).* In a matrix M for finding column minima, the element $M_{k,k'} = S_{PI_k, k'-k}$, for $0 \le k \le |PI| - 1$ and $k \le k' \le k + L$ if such $S_{PI_k, k'-k}$ exists.

With M, we have $PO_{k'} = \min_{0 \le k \le |PI|-1}\{M_{k,k'}\}$, for $0 \le k' \le L$. An example of M is shown in Table 3. The computation of PO is equivalent to finding the minimum of each column in M.

To find the column minima of M, the brute-force method needs $O(L^2)$ time to examine all the numbers, where L denotes the length of $LCS(A^{(r-1)}A^{(r)}, B)$. Note that $A^{(r-1)}A^{(r)}$ means the string concatenation of $A^{(r-1)}$ and $A^{(r)}$. Matrix M has been proved to be a *totally monotone matrix* [29], where the row minima do not occur in both the upper-right and lower-left corners for each 2 × 2 submatrix in M. Therefore, a recursive algorithm, named SMAWK and proposed by Aggarwal et al. [34], can find the column minima of a totally monotone matrix in $O(L)$ time [29]. With the S-table $S^{(r)}$ and the input PI, the alignment result PO of $G^{(r)}$ can be computed in $O(L)$ time, instead of the original DP approach in $O(mn)$ time.

In summary, for two substrings $A^{(1)}$, $A^{(2)}$ and one string B, the *concatenated LCS* (CoLCS) problem is to find the LCS length of $A^{(1)}A^{(2)}$ and B. When given the alignment result PI of $A^{(1)}$ and B, as well as the S-table $S^{(2)}$ of $A^{(2)}$ and B, it can be solved in $O(L)$ time [29], where the S-table is a 2-D matrix.

2.4. Merging of two S-tables

Here, we discuss the technique of merging two S-tables. Suppose $A^{(1)} = \texttt{cgga}$, $A^{(2)} = \texttt{ttct}$ and $B = \texttt{tctgatggt}$. The S-table of $A^{(2)}$ and B, denoted as $S^{(2)}$, is already shown in Table 2 and we show it again in the middle of Fig. 3. The S-table of $A^{(1)}A^{(2)}$ and B, denoted as $S^{(1+2)}$, is shown in the right of Fig. 3.

Row 0 of $S^{(1+2)}$ is the alignment of $A^{(1)}A^{(2)}$ and B, which is the same as the column minima in Table 3. Therefore, row 0 of $S^{(1+2)}$ can be computed with $S^{(2)}$ and row 0 of $S^{(1)}$. Also, row 1 of $S^{(1+2)}$ can be computed with $S^{(2)}$ and row 1 of $S^{(1)}$. That is, $S^{(1+2)}_i$ can be computed with $S^{(2)}$ and $S^{(1)}_i$ for $0 \le i \le n = |B|$, where $S^{(1)}_i$ denotes row i of $S^{(1)}$. In this way, computing the merged S-table $S^{(1+2)}$ needs $O(nL)$ time.

$S^{(1)}$	0	1	2	3
0	0	2	4	5
1	1	2	4	5
2	2	4	5	
3	3	4	5	
4	4	5	8	
5	5	7	8	
6	6	7	8	
7	7	8		
8	8			
9	9			

$S^{(2)}$	0	1	2	3
0	0	1	2	3
1	1	2	3	9
2	2	3	6	9
3	3	6	9	
4	4	6	9	
5	5	6	9	
6	6	9		
7	7	9		
8	8	9		
9	9			

$S^{(1+2)}$	0	1	2	3	4	5
0	0	1	2	3	6	9
1	1	2	3	5	6	9
2	2	3	5	6	9	
3	3	4	5	6	9	
4	4	5	6	9		
5	5	6	8	9		
6	6	7	8	9		
7	7	8	9			
8	8	9				
9	9					

Fig. 3. The S-tables. The left one is the S-table of $(A^{(1)}, B)$, the middle one is the S-table of $(A^{(2)}, B)$ and the right one is the S-table of $(A^{(1)}A^{(2)}, B)$, where $A^{(1)} = \texttt{cgga}$, $A^{(2)} = \texttt{ttct}$ and $B = \texttt{tctgatggt}$.

Table 4
The modified S-table and D with $A = \texttt{ttct}$ and $B = \texttt{tctgatggt}$, where the starting index i indicates $B_{i+1..n}$, and each element d_i in column D means that the number is the first occurrence in row i.

		Length k				
S-table		0	1	2	3	D
Starting index i	0	0	1	2	3	
	1	1	2	3	$\underline{9}$	$d_1 = 9$
	2	2	3	$\underline{6}$	9	$d_2 = 6$
	3	3	6	9	$\underline{\infty_1}$	$d_3 = \infty_1$
	4	$\underline{4}$	6	9	∞_1	$d_4 = 4$
	5	$\underline{5}$	6	9	∞_1	$d_5 = 5$
	6	6	9	∞_1	$\underline{\infty_2}$	$d_6 = \infty_2$
	7	$\underline{7}$	9	∞_1	∞_2	$d_7 = 7$
	8	$\underline{8}$	9	∞_1	∞_2	$d_8 = 8$
	9	9	∞_1	∞_2	$\underline{\infty_3}$	$d_9 = \infty_3$

3. An O($n \log n$)-time CoLCS algorithm with the linear-space S-table

For easy explanation, we translate ∞ into ∞_1, ∞_2, $\cdots$. Therefore, Table 2 is modified and shown in Table 4. In implementation, because $|B| = n$, we can regard ∞_1, ∞_2 and ∞_3 as $n+1$, $n+2$ and $n+3$, respectively.

The modified computation of matrix M is shown in Table 5(a). Clearly, the same result is obtained if only the finite values are considered when the column minima in M are computed. The finite values are considered as the output. The minimum of column 6 is ∞_1, so we can ignore it. The output of Table 5(a) is identical to Table 3.

Property 1. *Once a number x appears in $S_{i,*}$, x must appear in $S_{j,*}$ for $i \leq j \leq x$.*

Proof. This property can be derived from Theorem 1. □

Definition 8 *($C_{k,k'}$ and h_k).* Let $C_{k,k'}$ denote the cumulative minimum of $M_{0..k,k'}$, for $0 \leq k \leq |PI| - 1$ and $0 \leq k' \leq k + L$. Let hinge h_k denote the maximum of $M_{k,*} \setminus C_{k-1,*}$ ($M_{k,*}$ with excluding $C_{k-1,*}$), where the symbol $\setminus$ denotes the set difference operation.

For example, the matrix C is shown in Table 5(b) with $h_1 = 9$, $h_2 = \infty_1$, and $h_3 = 6$. By Definition 8, the alignment result $PO = C_{|PI|-1,*} = C_{3,*} = \langle 0, 1, 2, 3, 6, 9, \infty_1 \rangle$. We present a property of two consecutive $C_{k-1,*}$ and $C_{k,*}$ in Theorem 2.

Theorem 2. *$C_{k-1,*} \cup \{h_k\} = C_{k,*}$, for $1 \leq k \leq |PI| - 1$.*

Proof. Let y be the smallest index for $C_{k-1,y} > M_{k,y}$; otherwise, $y = |C_{k-1,*}|$. We can divide $C_{k,*}$ into two parts by index y as follows.

Table 5
The modified matrices M and C, where $PI = \langle 0, 2, 4, 5 \rangle$. (a) The matrix M, where each number in the bottom is the column minimum. (b) The matrix C.

		Length k'						
M		0	1	2	3	4	5	6
Row index k	0	0	1	2	3			
	1		2	3	6	9		
	2			4	6	9	∞_1	
	3				5	6	9	∞_1
Minimum		0	1	2	3	6	9	∞_1

(a)

		Length k'						
C		0	1	2	3	4	5	6
Row index k	0	0	1	2	3			
	1	0	1	2	3	9		
	2	0	1	2	3	9	∞_1	
	3	0	1	2	3	6	9	∞_1

(b)

1. $0 \le k' \le y-1$. In this case, $C_{k-1,k'} \le M_{k,k'}$. Thus, $C_{k,k'} = \min\{C_{k-1,k'}, M_{k,k'}\} = C_{k-1,k'}$.
2. $y \le k' \le k+L$. Because $C_{k-1,y} > M_{k,y}$, we have $C_{k,y} = \min\{C_{k-1,y}, M_{k,y}\} = M_{k,y}$. The value of $C_{k-1,k'}$ comes from one number in rows $PI_0, PI_1, \cdots, PI_{k-1}$ of S. $C_{k-1,*}$ is strictly increasing, $C_{k-1,k'} > M_{k,k'} \ge PI_k$, and $C_{k-1,k'}$ is some $M_{x,k'} = S_{PI_x,k'-x}$, so we can use Property 1. Except $h_k = M_{k,y}$, $C_{k-1,k'}$ appears in $M_{k,*}$ for all k'. Therefore, $C_{k,k'+1} = C_{k-1,k'}$.

Thus, $C_{k-1,*} \cup \{h_k\} = C_{k,*}$, where $h_k = M_{k,y}$. □

For example in Table 5, by treating each row as a set, $C_{1,*} = \{0, 1, 2, 3, 9\} = \{0, 1, 2, 3\} \cup \{9\} = C_{0,*} \cup \{9\}$, where $C_{0,*} = S_{0,*}$. $C_{2,*} = \{0, 1, 2, 3, 9\} \cup \{\infty_1\}$ and $C_{3,*} = C_{2,*} \cup \{6\}$, where 6 is the maximum of $M_{3,*} \setminus C_{2,*}$.

With the above properties and $PO = C_{|PI|-1,*}$, we can compute PO from $C_{0,*} = S_{0,*}$ sequentially. The CoLCS result PO consists of $S_{0,*}$ and the hinge values $H_{1..k} = \{h_1, h_2, \cdots, h_k\}$, where $k = |PI| - 1$. Since $S_{0,*}$ has already been given, we focus on finding $H_{1..k}$.

We find the value of h_k for $k = 1$ to $|PI| - 1$ sequentially.

Lemma 1. *$S_{i,*}$ consists of the L largest numbers in $S_{0,*} \cup D_{1..i}$, where $|S_{i,*}| = L$.*

Proof. Each element in $S_{i,*}$ is greater than or equal to i. By Theorem 1, the smallest number of $S_{i-1,*}$ does not appear in $S_{i,*}$. Then, the lemma holds. □

For example, in Table 4, $S_{2,*}$ consists of the four largest numbers $\{2, 3, 6, 9\}$ in $\{0, 1, 2, 3\} \cup \{9, 6\}$.

Theorem 3. *$h_k = \max(D_{1..PI_k} \setminus H_{1..k-1})$, for $1 \le k \le |PI| - 1$.*

Proof. By Definition 7, $M_{k,k'} = S_{PI_k,k'-k}$. With ignoring the detailed column index and regarding the numbers as a set, we have $M_{k,*} = S_{PI_k,*}$. By Definition 8 and Lemma 1, h_k is the maximum of $(S_{0,*} \cup D_{1..PI_k}) \setminus C_{k-1,*}$. Since $C_{0,*} = S_{0,*} \subseteq C_{k-1,*}$, we have that h_k is the maximum of $D_{1..PI_k} \setminus C_{k-1,*}$. By Theorem 2, $C_{k-1,*} \cup \{h_k\} = C_{k,*}$, so $C_{k-1,*} = C_{0,*} \cup \{h_1\} \cup \{h_2\} \cup \cdots \cup \{h_{k-1}\}$. We get Equation (5).

$$\begin{aligned} h_k &= \max(D_{1..PI_k} \setminus \{h_1, h_2, \cdots, h_{k-1}\}) \\ &= \max(D_{1..PI_k} \setminus H_{1..k-1}). \quad \square \end{aligned} \tag{5}$$

By Theorem 3, we can use a sequential method to find h_k by querying the range maximum of D, and to remove h_k from D after finding. Take Tables 4 and 5 as an example, where $PI = \langle 0, 2, 4, 5 \rangle$ and $M_{0,*} = \langle 0, 1, 2, 3 \rangle$. The sequential process is described as follows.

(1) $PI_1 = 2$, so we find h_1 in $\max(D_{1..2}) = 9$, and remove 9.
(2) $PI_2 = 4$ and $\max(D_{1..4}) = \infty_1$, so $h_2 = \infty_1$ and we remove ∞_1.
(3) $PI_3 = 5$ and $\max(D_{1..5}) = 6$, so $h_3 = 6$. Therefore, $PO = C_{3,*} = \langle 0, 1, 2, 3, 6, 9, \infty_1 \rangle$.

The range maximum query (RMQ) and single point update (removal) of D with the segment tree structure requires $O(\log n)$ time for each operation [35]. Thus, finding the CoLCS of $A^{(r-1)}A^{(r)}$ and B needs $O(|PI| \log n) = O(n \log n)$ time,

Table 6
An example of $nextPI$. If $nextPI(i)$ does not exist, we set it as empty. $PI = \langle 0, 2, 4, 5\rangle$ is underlined in the column (i).

i	d_i	$nextPI(i)$
1	9	2
$\underline{2}$	6	2
3	∞_1	4
$\underline{4}$	4	4
$\underline{5}$	5	5
6	∞_2	
7	7	
8	8	
9	∞_3	

when the linear-space S-table is given, including PI, $S^{(r)}_{0,*}$ (row 0 of S-table of $A^{(r)}$ and B) and $D^{(r)}$. The algorithm is presented in Algorithm 1.

Algorithm 1 An O($n \log n$)-time CoLCS algorithm.

```
Input: $PI$, $S_{0,*}$ and $D$
Output: $PO$
$PO \leftarrow S_{0,*}$                                  ▷ insert $S_{0,*}$ into $PO$
for $k = 1$ to $|PI| - 1$ do
    $i \leftarrow \arg\max(D_{1..PI_k})$                 ▷ $i$ is the index of the range maximum
    insert $d_i$ into $PO$                               ▷ $h_k = d_i$
    $d_i \leftarrow -\infty$                             ▷ remove $d_i$ from $D$
return $PO$
```

4. An O(n)-time CoLCS algorithm with the linear-space S-table

In this section, we present a more efficient way for obtaining h_k. The process involved in Algorithm 1 is a sequence of RMQ's and maximum deletions. By using the concept of union-find, Eppstein and Muthukrishnan [36] proposed an offline priority queue to answer the range maxima in a sequence of element insertions and deletions. In their problem, the life time of each element is known in the offline process. However, the life time of each element in our problem is not known in advance even in the offline process, because only after a range maximum has been found, the termination (right boundary) of that maximum element is decided. Though we cannot directly apply the offline priority queue to solving our problem, we can still use the similar concept of union-find to improve the efficiency of our algorithm.

Now we focus on whether the number d_i, with the decreasing order, will become the value of a certain h_k or not. We stipulate that $\infty_i < \infty_{i'}$ if $i < i'$.

Definition 9 *($nextPI(i)$).* For d_i, let $nextPI(i)$ be the smallest PI_k such that $i \leq PI_k$.

The $nextPI$ of Table 4 is shown in Table 6, where $PI = \langle 0, 2, 4, 5\rangle$. Note $nextPI(i)$ indicates the potential right boundary for d_i to be an answer in an RMQ.

When we examine d_i, we use the *union-find* data structure [37,38] to check whether PI_k and h_k have been examined or not. If PI_k and h_k have been examined, we have to try the next, PI_{k+1} and h_{k+1}. The operations in the union-find data structure are listed as follows.

- $make(x, C)$: Create a new set named C containing exactly x.
- $find(x)$: Find the name of the set containing x.
- $union(x, y, C)$: Unite the set containing x and the set containing y into a new set named C.

In the union-find data structure, each number in PI except PI_0 is initially in a unique set, implemented by $make(PI_k, k)$ for $1 \leq k \leq |PI| - 1$. We also use $make(\infty, |PI|)$ to set the boundary. Our algorithm for finding PO is presented in Algorithm 2, where D is sorted in decreasing order.

Fig. 4 shows an example of the union-find process, with $PI = \langle 0, 2, 4, 5\rangle$ and $D = \langle 9, 6, \infty_1, 4, 5, \infty_2, 7, 8, \infty_3\rangle$. In this case, $|PI| = 4$ is the boundary number. We check D with the decreasing order, shown as follows.

1. For $d_9 = \infty_3$ and $d_6 = \infty_2$, since $nextPI(9)$ and $nextPI(6)$ are empty, meaning they are not answers of any RMQ, so skip them.
2. For $d_3 = \infty_1$, we have $nextPI(3) = PI_2 = 4$, and $k = find(4) = 2 \neq |PI| = 4$, meaning $h_2 = \max(D_{1..PI_2}) = \max(D_{1..4}) = \infty_1$. So we have $union(PI_2, PI_3, find(PI_3)) = union(PI_2, PI_3, 3)$. In this situation, h_2 with PI_2 has been examined. If

Algorithm 2 An O(n)-time CoLCS algorithm.

```
Input: $PI$, $S_{0,*}$, $D$ and $nextPI$, where $D$ is sorted in decreasing
Output: $PO$
$PO \leftarrow S_{0,*}$                                         ▷ insert each of $S_{0,*}$ into $PO$
for $k = 1$ to $|PI| - 1$ do
    $make(PI_k, k)$
$make(\infty, |PI|)$                                            ▷ set the boundary
for $d_i \in D$ from the largest to the smallest number do      ▷ decreasing order, achieved by bucket sort
    if $nextPI(i)$ exists and $find(nextPI(i)) \neq |PI|$ then
        set $k \leftarrow find(nextPI(i))$
        insert $d_i$ into $PO$                                  ▷ $h_k = d_i$
        $union(PI_k, PI_{k+1}, find(PI_{k+1}))$
return $PO$
```

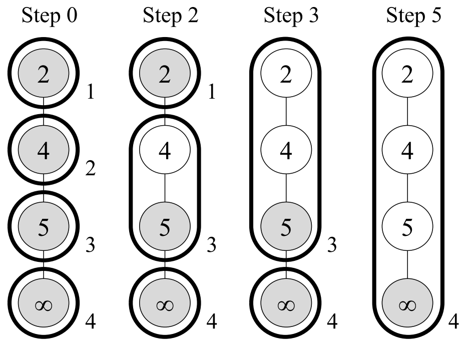


Fig. 4. An example of the union-find process. Each thin circle is an element, and each bold circle is a set. The number beside each set is the name of the set.

h_2 is desired to be set next time, $union(PI_2, PI_3, 3)$ guarantees to set h_3 with PI_3, instead of h_2. In other words, when either h_2 or h_3 may be set next time, we always set h_3.

3. For $d_1 = 9$, we have $nextPI(1) = PI_1 = 2$, and $k = find(2) = 1 \neq |PI| = 4$, meaning $h_1 = \max(D_{1..PI_1}) = \max(D_{1..2}) = 9$. We do $union(PI_1, PI_2, find(PI_2)) = union(PI_1, PI_2, 3)$. After $union(PI_1, PI_2, 3)$ is performed, if one of h_1, h_2 and h_3 is desired to be set next time, we always set h_3.
4. $nextPI(8)$ and $nextPI(7)$ are empty, so skip them.
5. For $d_2 = 6$, we have $nextPI(2) = 2$ and $k = find(2) = 3 \neq |PI| = 4$. It means meaning $h_3 = \max(D_{1..PI_3}) = \max(D_{1..5}) = 6$ (excluding $d_3 = \infty_1$ and $d_1 = 9$). Then $union(PI_3, PI_4, find(PI_4)) = union(PI_3, PI_4, 4)$.
6. The algorithm finishes after $H_{1..3}$ are found. If we check the next number $d_5 = 5$, $nextPI(5) = PI_3 = 5$, and $find(5) = |PI| = 4$. $d_5 = 5$ cannot be the value of any h_k.

Finally, output $PO = S_{0,*} \cup H_{1..3} = \langle 0, 1, 2, 3\rangle \cup \langle 9, \infty_1, 6\rangle = \langle 0, 1, 2, 3, 6, 9, \infty_1\rangle$.

For the general union-find problem, the time required for each operation of union or finding is O($\alpha(n)$), where the lower bound of $\alpha(n)$ was proved to be functional inverse of Ackermann's function [38]. The union-find structure we use is a single path tree, and we only unite two consecutive sets. With the definition of static tree set union, the time complexity of each operation is reduced to O(1) [37]. Our algorithm needs O(n) operations of union-find, so the time complexity is O(n). The CoLCS length can be computed in linear time, when the linear-space S-table is given. In order to get PO with increasing order, we can collect the elements h_k, and apply the bucket sort on these elements with an array of size n. It needs O(n) time. In summary, the CoLCS problem with the linear-space S-table can be solved in linear time.

5. Merging two linear-space S-tables in O($n \log n$) time

In this section, we propose an O($n \log n$)-time algorithm to merge two linear-space S-tables. Fig. 5 shows three S-tables, where the right one is the merged result of the left two. For easy explanation, we fill in the infinity symbols in the middle table and use $S^{(3)}$ to denote $S^{(1+2)}$. Our goal is to calculate the first row of $S^{(3)}$ ($S^{(3)}_{0,*}$) and $D^{(3)}$ with the linear-space S-tables ($S^{(1)}_{0,*}$, $D^{(1)}$, $S^{(2)}_{0,*}$ and $D^{(2)}$). Since in Section 4, $S^{(3)}_{0,*} = PO$ can be computed in O(n) time, in this section, we focus on the computation of $D^{(3)}$ in O($n \log n$) time.

In Section 3, the main idea for calculating $PO = S^{(3)}_{0,*}$ is to find the range maximum in $D_{1..PI_k} \setminus H_{1..k-1}$ repeatedly, for $1 \leq k \leq |PI| - 1$. Now, we apply the same concept for computing $D^{(3)}$. In order to clearly explain the calculation of each row $S^{(3)}_{i,*}$, corresponding to $B_{i+1..n}$, we denote C, H, M, PI and PO as C^i, H^i, M^i, PI^i and PO^i, respectively. By the definition

$S^{(1)}$	0	1	2	3	$D^{(1)}$
0	0	2	4	5	
1	1	2	4	5	1
2	2	4	5		∞
3	3	4	5		3
4	4	5	8		8
5	5	7	8		7
6	6	7	8		6
7	7	8			∞
8	8				∞
9	9				9

$S^{(2)}$	0	1	2	3	$D^{(2)}$
0	0	1	2	3	
1	1	2	3	9	9
2	2	3	6	9	6
3	3	6	9	∞_1	∞_1
4	4	6	9	∞_1	4
5	5	6	9	∞_1	5
6	6	9	∞_1	∞_2	∞_2
7	7	9	∞_1	∞_2	7
8	8	9	∞_1	∞_2	8
9	9	∞_1	∞_2	∞_3	∞_3

$S^{(3)}$	0	1	2	3	4	5	$D^{(3)}$
0	0	1	2	3	6	9	
1	1	2	3	5	6	9	5
2	2	3	5	6	9		∞
3	3	4	5	6	9		4
4	4	5	6	9			∞
5	5	6	8	9			8
6	6	7	8	9			7
7	7	8	9				∞
8	8	9					∞
9	9						∞

Fig. 5. Three S-tables. The left $S^{(1)}$ is the S-table of $(A^{(1)}, B)$, the middle $S^{(2)}$ is the S-table of $(A^{(2)}, B)$ and the right $S^{(3)}$ is the S-table of $(A^{(1)}A^{(2)}, B)$, where $A^{(1)} = \texttt{cgga}$, $A^{(2)} = \texttt{ttct}$ and $B = \texttt{tctgatggt}$.

M^0	0	1	2	3	4	5	6
0	0	1	2	3			
1		2	3	6	$\underline{9}$		
2			4	6	9	$\underline{\infty_1}$	
3				5	$\underline{6}$	9	∞_1
	0	1	2	3	6	9	∞_1

M^1	0	1	2	3	4	5	6
0	1	2	3	9			
1		2	3	$\underline{6}$	9		
2			4	6	9	$\underline{\infty_1}$	
3				$\underline{5}$	6	9	∞_1
	1	2	3	5	6	9	∞_1

M^2	0	1	2	3	4	5
0	2	3	6	9		
1		4	6	9	$\underline{\infty_1}$	
2			$\underline{5}$	6	9	∞_1
	2	3	5	6	9	∞_1

C^0	0	1	2	3	4	5	6
0	0	1	2	3			
1	0	1	2	3	$\underline{9}$		
2	0	1	2	3	9	$\underline{\infty_1}$	
3	0	1	2	3	$\underline{6}$	9	∞_1

C^1	0	1	2	3	4	5	6
0	1	2	3	9			
1	1	2	3	$\underline{6}$	9		
2	1	2	3	6	9	$\underline{\infty_1}$	
3	1	2	3	$\underline{5}$	6	9	∞_1

C^2	0	1	2	3	4	5
0	2	3	6	9		
1	2	3	6	9	$\underline{\infty_1}$	
2	2	3	$\underline{5}$	6	9	∞_1

Fig. 6. M^0, C^0, M^1, C^1, M^2 and C^2 for $A^{(1)} = \texttt{cgga}$, $A^{(2)} = \texttt{ttct}$ and $B = \texttt{tctgatggt}$. The grey numbers in M^0 become $M^1_{0,*}$. The underline numbers are selected into H. The red circle shows where the number $d^{(3)}_1 = 5$ comes from. Here, $PI^0 = S^{(1)}_{0,*} = \langle 0, 2, 4, 5\rangle$, $PI^1 = S^{(1)}_{1,*} = \langle 1, 2, 4, 5\rangle$, $H^0 = \langle 9, \infty_1, 6\rangle$ and $H^1 = \langle 6, \infty_1, 5\rangle$.

of CoLCS, $PI^i = S^{(1)}_{i,*}$ and $PO^i = S^{(3)}_{i,*}$, where $S^{(3)}_{i,*} = S^{(2)}_{i,*} \cup H^i$. In fact, we do not really compute the whole row $S^{(3)}_{i,*}$. By discussing the properties in C^i, H^i and M^i, we can efficiently get $d^{(3)}_i$.

Fig. 6 shows an example of the change between C^0 and C^1, and the change between M^0 and M^1. To compute $d^{(3)}_1$, we observe the difference between $C^0_{3,*}$ and $C^1_{3,*}$. The red circle 5 is the number that appears in $C^1_{3,*}$ but not in $C^0_{3,*}$. Therefore, we have $h^1_3 = d^{(3)}_1 = 5$. The way for getting $d^{(3)}_1 = 5$ is described more precisely in the following.

In Theorem 3, the computation is for h^0_k. We can easily extend it to compute h^i_k.

Corollary 1. *$h^i_k = \max(D^{(2)}_{i+1..PI^i_k} \setminus H^i_{1..k-1})$, for $1 \le k \le |PI^i| - 1$.*

$PI^0 = S^{(1)}_{0,*} = \langle 0, 2, 4, 5\rangle$ and $PI^1 = S^{(1)}_{1,*} = \langle 1, 2, 4, 5\rangle$. The numbers of h^i_k are underlined in Fig. 6. We have already $h^0_1 = 9$, $h^0_2 = \infty_1$ and $h^0_3 = 6$ in C^0. 9 appears in $M^1_{0,*}$, so we find a new range maximum in $D^{(2)}$ for h^1_1. That is, we get $h^1_1 = \max(D^{(2)}_{2..2} \setminus \emptyset) = 6$. Next, we have $h^1_2 = \max(D^{(2)}_{2..4} \setminus \{6\}) = \infty_1$ and $h^1_3 = \max(D^{(2)}_{2..5} \setminus \{6, \infty_1\}) = 5$. As we can see that some h^0_k may become other $h^1_{k'}$. The new appearance $d^{(3)}_1$ is the final range maximum $d^{(3)}_1 = h^1_3 = \max(D^{(2)}_{2..5} \setminus \{6, \infty_1\}) = 5$. Thus, $H^0 = \langle 9, \infty_1, 6\rangle$ and $H^1 = \langle 6, \infty_1, 5\rangle$. The changes of the h^i_k values are shown in Fig. 7.

For computing $d^{(3)}_i$, we consider $d^{(1)}_i = \infty$ and $d^{(1)}_i \neq \infty$ in Theorems 4 and 5, respectively. In addition, Theorem 5 discusses finite $d^{(1)}_i = i$ and $d^{(1)}_i \neq i$ in Case 1 and Case 2, separately. An example for the computation will be given in details after Theorem 5.

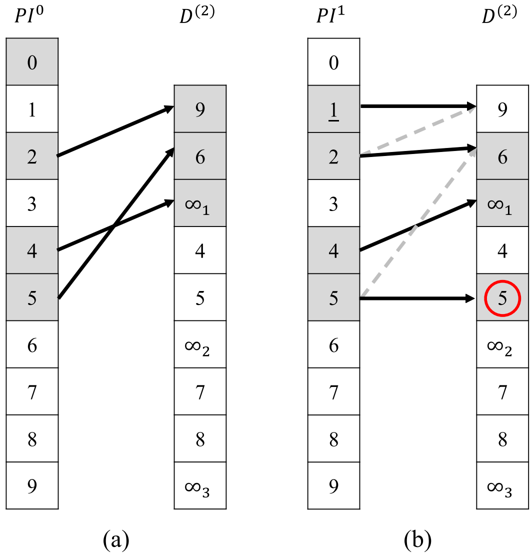


Fig. 7. The changes of the h^i_k values from $PI^0 = S^{(1)}_{0,*} = \langle 0, 2, 4, 5\rangle$ to $PI^1 = S^{(1)}_{1,*} = \langle 1, 2, 4, 5\rangle$. (a) The sources of $H^0 = \langle 9, \infty_1, 6\rangle$ in $D^{(2)}$. (b) The sources of $H^1 = \langle 6, \infty_1, 5\rangle$ in $D^{(2)}$.

Theorem 4. *$d^{(3)}_i = \infty$ if $d^{(1)}_i = \infty$, for $1 \le i \le n$.*

Proof. By Definition 7, $M^i_{0,*} = S^{(2)}_{i,*}$. If $d^{(1)}_i = \infty$, then $M^i_{j,*} = S^{(2)}_{PI^i_j,*} = S^{(2)}_{PI^{i-1}_{j+1},*} = M^{i-1}_{j+1,*}$. That is, M^i can be obtained from M^{i-1} by removing row 0 and shifting the elements with one left position. Therefore, we have $C^i_{j,*} = C^{i-1}_{j+1,*} \setminus \{i-1\}$. In other words, $S^{(3)}_{i,*} = S^{(3)}_{i-1,*} \setminus \{i-1\}$. Thus, $d^{(3)}_i = \infty$. □

Definition 10 *($\delta^i(i')$ and balance point y).* Let $\delta^i(i') = |D^{(2)}_{1..i'} \cap H^i| - |\{x | x \in PI^i, i+1 \le x \le i'\}|$. If there exists the smallest index $y \ge d^{(1)}_i$ such that $\delta^{(i-1)}(y) = 0$, y is the balance point for finding $d^{(3)}_i$.

The difference $\delta^i(i')$ indicates whether there are enough feasible elements in PI^i to cover the current $H^i \cap D^{(2)}_{i+1..i'}$, that is, $h_k \in S^{(2)}_{x,*} = M^i_{k,*}$ for some $x \in PI^i_{i+1..i'}$. The balance point y helps to efficiently find a newer $h_k = d^{(3)}_i$ from H^{i-1} to H^i.

For example (see Fig. 5, 6 and Table 7), $D^{(2)} = \langle 9, 6, \infty_1, 4, 5, \infty_2, 7, 8, \infty_3\rangle$, $H^0 = \langle 9, \infty_1, 6\rangle$ and $PI^0 = \langle 0, 2, 4, 5\rangle$. We have $\delta^0(1) = |\{9\} \cap \{9, \infty_1, 6\}| - |\emptyset| = 1$, $\delta^0(4) = |\{9, 6, \infty_1, 4\} \cap \{9, \infty_1, 6\}| - |\{2, 4\}| = 1$, and $\delta^0(5) = |\{9, 6, \infty_1, 4, 5\} \cap \{9, \infty_1, 6\}| - |\{2, 4, 5\}| = 0$. In summary, $\delta^0 = \langle 1, 1, 2, 1, 0, 0, 0, 0, 0\rangle$. $\delta^0(4) = 1$ means that h^1_2 can still be found in $\{9, \infty_1, 6\} \setminus \{9\}$ after 9 is shifted to become one member of $S^{(3)}_{0,*}$. However, $\delta^0(5) = 0$ means that $h^1_3 = d^{(3)}_1$ is not in $\{9, \infty_1, 6\} \setminus \{9\}$, since $\{\infty_1, 6\}$ is for h^1_1 and h^1_2. We have to perform $d^{(3)}_1 = h^1_3 = \max(D^{(2)}_{1..5} \setminus H^0) = 5$. For $i = 1$, to find $d^{(3)}_i = d^{(3)}_1$, the smallest position $y = 5$ ($y \ge d^{(1)}_i$) is the balance point that $\delta^{(i-1)}(y) = \delta^0(5) = 0$.

With Corollary 1, we have $\delta^i(i') \ge 0$. Clearly, $\delta^i(PI^i_{|PI^i|-1}) = 0$ is true. For example, $\delta^0(PI^0_3 = 5) = 0$ and $\delta^4(PI^4_2 = 8) = 0$. Thus, there exists some y such that $\delta^i(y) = 0$. Except $d^{(1)}_i = d^{(3)}_i = \infty$ in Theorem 4, the smallest $y \ge d^{(1)}_i$ such that $\delta^{i-1}(y) = 0$ is the key position for determining $d^{(3)}_i$, described in the following theorem.

Theorem 5. *If $d^{(1)}_i \ne \infty$ and y is the smallest index such that $y \ge d^{(1)}_i$ and $\delta^{i-1}(y) = 0$, then $d^{(3)}_i = \max(D^{(2)}_{i..y} \setminus H^{i-1})$.*

Proof. Two cases $d^{(1)}_i = i$ and $d^{(1)}_i \ne i$ are considered as follows.

1. $d^{(1)}_i = i$. In this case, we have $PI^i_0 = S^{(1)}_{i,0} = i$ and $S^{(1)}_{i,k} = PI^i_k = PI^{i-1}_k = S^{(1)}_{i-1,k}$, for $1 \le k \le |PI^i| - 1$. By Definition 6, $d^{(2)}_i \in S^{(2)}_{i,*} = C^i_{0,*}$, we further consider two subcases for $d^{(2)}_i$ as follows.
   (a) $d^{(2)}_i \notin H^{i-1}$. Because $d^{(2)}_i \notin H^{i-1}$, $d^{(2)}_i \notin S^{(2)}_{i-1,*}$ and $d^{(2)}_i \in S^{(2)}_{i,*}$, the new appearance $d^{(3)}_i = d^{(2)}_i$. Now, we want to prove that $\delta^{i-1}(i) = 0$. By Corollary 1, $H^{i-1} \subseteq \{d^{(2)}_i, d^{(2)}_{i+1}, \ldots\}$. Since $d^{(2)}_i \notin H^{i-1}$, we have $D^{(2)}_{1..i} \cap H^{i-1} = \emptyset$. $d^{(1)}_i = i$, so

Table 7
The δ^{i-1} values and H^{i-1} for finding $d_i^{(3)}$. The bold red numbers mean that the range update happens in δ by Theorem 5. '-' in the y value means no available y value, since by Theorem 4, $d_i^{(3)} = \infty$ if $d_i^{(1)} = \infty$.

		$D^{(1)}$	$D^{(2)}$	$D^{(3)}$	δ^0	δ^1	δ^2	δ^3	δ^4	δ^5	δ^6	δ^7	δ^8	δ^9
y					5	–	4	8	8	7	–	–	9	–
i	1	1	9	5	1	**0**	0	0	0	0	0	0	0	0
	2	∞	6	∞	1	**0**	0	0	0	0	0	0	0	0
	3	3	∞_1	4	2	**1**	1	**0**	0	0	0	0	0	0
	4	8	4	∞	1	**0**	0	0	0	0	0	0	0	0
	5	7	5	8	0	0	0	0	0	0	0	0	0	0
	6	6	∞_2	7	0	0	0	0	**1**	1	**0**	0	0	0
	7	∞	7	∞	0	0	0	0	**1**	**0**	0	0	0	0
	8	∞	8	∞	0	0	0	0	0	0	0	0	0	0
	9	9	∞_3	∞	0	0	0	0	0	0	0	0	0	0
H	1				9	6	∞_1	4	5	∞_2	7	8	–	–
	2				∞_1	∞_1	5	5	∞_2	8	8	–	–	–
	3				6	5	–	–	–	–	–	–	–	–

$i \notin PI^{i-1}$, and then $\{x|x \in PI^{i-1}, i \le x \le i\} = \emptyset$. Therefore, by Definition 10, $\delta^{i-1}(i) = |D^{(2)}_{1..i} \cap H^{i-1}| - |\{x|x \in PI^{i-1}, i \le x \le i\}| = 0 - 0 = 0$. Finally, we get $d_i^{(3)} = \max(D^{(2)}_{i..i} \setminus H^{i-1}) = d_i^{(2)}$.

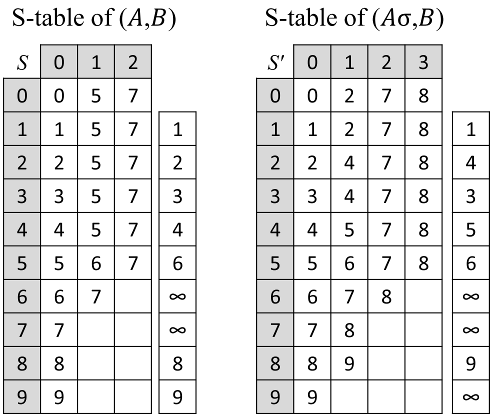
(b) $d_i^{(2)} \in H^{i-1}$. By Theorem 1, $S^{(2)}_{i,*} = S^{(2)}_{i-1,*} \cup \{d_i^{(2)}\} \setminus \{i-1\}$. By Theorem 2, $S^{(3)}_{i,*} = S^{(2)}_{i,*} \cup H^i = S^{(2)}_{i-1,*} \cup H^i \cup \{d_i^{(2)}\} \setminus \{i-1\}$. Then $S^{(3)}_{i,*} = S^{(3)}_{i-1,*} \cup \{d_i^{(3)}\} \setminus \{i-1\} = S^{(2)}_{i-1,*} \cup H^{i-1} \cup \{d_i^{(3)}\} \setminus \{i-1\}$. By Definition 6 and Corollary 1, $(S^{(2)}_{i-1,*} \cup \{i-1\}) \cap (H^i \cup \{d_i^{(2)}\}) = \emptyset$ and $(S^{(2)}_{i-1,*} \cup \{i-1\}) \cap (H^{i-1} \cup \{d_i^{(3)}\}) = \emptyset$, we have $H^i \cup \{d_i^{(2)}\} = H^{i-1} \cup \{d_i^{(3)}\}$. $d_i^{(2)} \in H^{i-1}$, so $\{d_i^{(3)}\} = H^i \setminus H^{i-1}$, and we also have $H^{i-1} \setminus \{d_i^{(2)}\} \subset H^i$. By Corollary 1, $H^i \subseteq \{d^{(2)}_{i+1}, d^{(2)}_{i+2}, \ldots\}$, so $d_i^{(2)} \notin H^i$, and all elements in H^{i-1} except $d_i^{(2)}$ will appear in H^i.

Suppose y is the smallest index such that $y \ge d_i^{(1)} = i$ and $\delta^{i-1}(y) = 0$, and suppose k is the index such that $PI_k^{i-1} = PI_k^i = y$. $d_i^{(2)} \notin H^i$ and $d_i^{(2)} \in H^{i-1}$, so $\delta^i(y)$ will become 0, meaning that $h_k^i \notin H^{i-1}$. If $h_k^i \in H^{i-1}$, then $\delta^{i-1}(y) \ge 1$ holds. Therefore, we have $d_i^{(3)} = H^i \setminus H^{i-1} = h_k^i = \max(D^{(2)}_{i..y} \setminus H^{i-1})$.

2. $d_i^{(1)} \ne i$. Suppose k is the index that $S^{(1)}_{i,k} = d_i^{(1)}$. Two subcases for the changes of PI^i and H^i are considered as follows.
   (a) $0 \le j \le k-1$. We have $PI_j^i = PI_{j+1}^{i-1}$. This situation is the same as Theorem 4, so $C^i_{j,*} = C^{i-1}_{j+1,*} \setminus \{i-1\}$. Then, $h^i_{j'} = h^{i-1}_{j'+1}$ for $1 \le j' \le k-1$ and the value of h_1^{i-1} appears in $C^i_{0,*}$. It needs to check for $j \ge k+1$.
   (b) $j \ge k+1$. We have $PI_j^i = PI_j^{i-1}$. This subcase is similar to subcase 1(b). Suppose y is the smallest index such that $y \ge d_i^{(1)}$ and $\delta^{i-1}(y) = 0$, and suppose k' is the index such that $PI^{i-1}_{k'} = PI^i_{k'} = y$. Because $d_i^{(1)}$ appears in PI^i, $\delta^i(y)$ will become 0. Therefore, we have $d_i^{(3)} = h^i_{k'} = \max(D^{(2)}_{i..y} \setminus H^{i-1})$. □

By Theorems 4 and 5, we can find $D^{(3)}$ using a sequential method, starting from $d_1^{(3)}$, then $d_2^{(3)}$, $d_3^{(3)}$, $\cdots$, and so on. Corresponding to $d_i^{(3)}$, the values in δ^i can be obtained by modifying δ^{i-1}. For deciding $d_i^{(3)}$, we need to find the smallest index $y \ge d_i^{(1)}$ such that $\delta^{i-1}(y) = 0$, which can be achieved by the range minimum finding on the 2-tuples $\langle \delta^{i-1}(j), j\rangle$, whose value is $\langle 0, y\rangle$. Then, we do a range maximum query $d_i^{(3)} = \max(D^{(2)}_{i..y} \setminus H^{i-1})$. After that, a range update of $\langle \delta^{i-1}(j), j\rangle$ can efficiently obtain δ^i for the next iteration. If $d_i^{(1)} \ne \infty$, because $d_i^{(1)}$ appears in PI^i and $PI_0^i = i$, we have $\delta^i(j) = \delta^{i-1}(j) + 1$, for $i \le j < d_i^{(1)}$. Suppose k is such that $d_k^{(2)} = d_i^{(3)}$. Because $d_k^{(2)}$ appears in H^i and i is used in $C^i_{0,*}$, we have $\delta^i(j) = \delta^{i-1}(j) - 1$, for $i \le j < k$.

The δ values for computing $D^{(3)}$ in Fig. 5 are shown in Table 7. In this example, $d_i^{(1)} = \infty$ for $i \in \{2, 7, 8\}$ applies Theorem 4; $d_i^{(1)} = i$ follows subcases 1(a) and 1(b) in Theorem 5 for $i = 9$ and $i \in \{1, 3, 6\}$, respectively; and $d_i^{(1)} \ne i$ follows case 2 in Theorem 5 for $i \in \{4, 5\}$. When $i = 1$, $y = 5 \ge d_i^{(1)}$ is obtained from $\delta^{i-1} = \delta^0$, and $d_i^{(3)} = \max(D^{(2)}_{i..y} \setminus H^{i-1}) = \max(\{9, 6, \infty_1, 4, 5\} \setminus \{9, \infty_1, 6\}) = 5$. Since $d_1^{(3)} = 5 = d^{(2)}_{k=5}$, we can obtain δ^1 from δ^0 by range update $\delta^i(j) = \delta^{i-1}(j) - 1$ for $i = 1 \le j < k = 5$. When $i = 2$, $d_i^{(1)} = \infty$ implies $d_i^{(3)} = \infty$ and δ^i is identical to δ^{i-1}. When $i = 3$, $y = 4$ is obtained from $\delta^{i-1} = \delta^2$, and $d_i^{(3)} = \max(D^{(2)}_{i..y} \setminus H^{i-1}) = \max(\{\infty_1, 4\} \setminus \{\infty_1, 5\}) = 4$. Since $d_3^{(3)} = 4 = d^{(2)}_{k=4}$, we can obtain δ^3 from δ^2 by range update $\delta^i(j) = \delta^{i-1}(j) - 1$ for $i = 3 \le j < k = 4$. When $i = 4$, $d_4^{(1)} = 8$, $y = 8 \ge d_i^{(1)}$ is obtained from $\delta^{i-1} = \delta^3$, and $d_i^{(3)} = \max(D^{(2)}_{i..y} \setminus H^{i-1}) = \max(\{4, 5, \infty_2, 7, 8\} \setminus \{4, 5\}) = \infty_2$. We then temporarily obtain $\langle 0, 0, 0, 1, 1, 1, 1, 0, 0\rangle$ by updating



S-table of (A,B)

S	0	1	2	
0	0	5	7	
1	1	5	7	1
2	2	5	7	2
3	3	5	7	3
4	4	5	7	4
5	5	6	7	6
6	6	7		∞
7	7			∞
8	8			8
9	9			9

S-table of $(A\sigma,B)$

S'	0	1	2	3	
0	0	2	7	8	
1	1	2	7	8	1
2	2	4	7	8	4
3	3	4	7	8	3
4	4	5	7	8	5
5	5	6	7	8	6
6	6	7	8		∞
7	7	8			∞
8	8	9			9
9	9				∞

Fig. 8. Two S-tables with $A = \texttt{taa}, \sigma = \texttt{g}$ and $B = \texttt{cgcgatagg}$. The left one is the S-table of (A, B), denoted by S, and the right one is the S-table of $(A\sigma, B)$, denoted by S'.

$\delta^i(j) = \delta^{i-1}(j) + 1$ for $i = 4 \le j < d_i^{(1)} = 8$. Since $d_4^{(3)} = \infty_2 = d_{k=6}^{(2)}$, it performs the range update by $\delta^i(j) = \delta^{i-1}(j) - 1$ for $i = 4 \le j < k = 6$ to obtain $\delta^4 = \langle 0, 0, 0, 0, 0, 1, 1, 0, 0 \rangle$.

The algorithm is shown in Algorithm 3. In Section 4, we compute $S_{0,*}^{(3)}$ in $O(n)$ time. For computing each $d_i^{(3)}$, we use two segment trees, one for maintaining the range minimum of δ and the other for maintaining the range maximum of $D^{(2)}$. We need to do range query and range update in the two segment trees, where $BT.update(i_1, i_2, c_1)$ updates values in a range such that $BT[i'] \leftarrow BT[i'] + c_1$ for $i_1 \le i' \le i_2$. Each operation of the segment tree costs $O(\log n)$ time [35]. Thus, the time complexity of merging two linear-space S-tables is $O(n \log n)$ time.

Algorithm 3 An $O(n \log n)$-time algorithm for merging two linear-space S-tables.

```
Input: $S_{0,*}^{(1)}$, $D^{(1)}$, $S_{0,*}^{(2)}$ and $D^{(2)}$
Output: $S_{0,*}^{(3)}$ and $D^{(3)}$
$BT$ is a segment tree of $\langle \delta^{i-1}(j), j \rangle$ for range minimum query
$DT$ is a segment tree which records range maximum of $D^{(2)}$
compute $S_{0,*}^{(3)}$ with Algorithm 2, and update $BT$ and $DT$
for $i = 1$ to $n$ do
    if $d_i^{(1)} = \infty$ then
        $d_i^{(3)} \leftarrow \infty$
    else
        $y \leftarrow BT.range\_min\_query(d_i^{(1)}, n)$                ▷ $\delta^{i-1}(y) = 0$
        $d_k^{(2)} \leftarrow DT.range\_max\_query(i, y)$
        $d_i^{(3)} \leftarrow d_k^{(2)}$
        $DT.update(k, k, -\infty)$                ▷ increase $-\infty$ to interval $[k, k]$, delete it
        if $d_i^{(1)} > k$ then
            $BT.update(k, d_i^{(1)} - 1, 1)$                ▷ increase 1 to interval $[k, d_i^{(1)} - 1]$
        else if $d_i^{(1)} < k$ then
            $BT.update(d_i^{(1)}, k - 1, -1)$                ▷ increase -1 to interval $[d_i^{(1)}, k - 1]$
```

6. An algorithm for building the linear-space S-table incrementally

In this section, given the linear-space S-table of A and B, we propose an algorithm for building the linear-space S-table of $A\sigma$ and B in $O(n)$ time, where σ is a new character appended at the tail of A. Fig. 8 shows an example of $(A, \sigma, B) = (\texttt{taa}, \texttt{g}, \texttt{cgcgatagg})$, where S denotes the S-table of A and B, and S' denotes the S-table of $A\sigma$ and B.

With the new character σ, the values in S' may be decreased since we may use fewer characters in sequence B to achieve the same LCS length. For example, $S_{0,1} = 5$ means $|LCS(\texttt{taa}, \texttt{cgcga})| = 1$, which uses the first 5 characters of B to achieve the LCS length 1, while $S'_{0,1} = 2$ means $|LCS(\texttt{taag}, \texttt{cg})| = 1$, which uses only the first 2 characters of B to achieve the same LCS length 1. We observe the changes of $S'_{0,*}$ first.

Definition 11 *(next(i)).* Let $next(i)$ be the smallest j in B such that $i < j$ and $b_j = \sigma$, and $next(i) = \infty$ if there is no such j.

For example, suppose $\sigma = \texttt{g}$ and $B = \texttt{cgcgatagg}$. $next(0) = 2$, $next(2) = 4$, and $next(5) = 8$. In Fig. 8, $next(S_{0,0}) = next(0) = 2 < S_{0,1} = 5$, then we get $S'_{0,1} = 2$. Accordingly, we have the following theorem.

Table 8
The replacement status array Z for transforming D to D' and the array *next*, where $A = \texttt{taa}, \sigma = \texttt{g}$ and $B = \texttt{cgcgatagg}$. Here, the values of Z in row d_i are the statuses after d'_i has been found out.

index j of B		0	1	2	3	4	5	6	7	8	9	10
initialization by $S_{0,*}$		O	U	U	U	U	O	U	O	U	U	O
then by $S'_{0,*}$		O	N	R	U	U	O	N	O	R	U	O
$d_1 = 1$	$d'_1 = 1$	O	O	R	U	U	O	N	O	R	U	O
$d_2 = 2$	$d'_2 = 4$	O	O	O	N	R	O	N	O	R	U	O
$d_3 = 3$	$d'_3 = 3$	O	O	O	O	R	O	N	O	R	U	O
$d_4 = 4$	$d'_4 = 5$	O	O	O	O	O	O	N	O	R	U	O
$d_5 = 6$	$d'_5 = 6$	O	O	O	O	O	O	O	O	R	U	O
$d_6 = \infty$	$d'_6 = \infty$	O	O	O	O	O	O	O	O	R	U	O
$d_7 = \infty$	$d'_7 = \infty$	O	O	O	O	O	O	O	O	R	U	O
$d_8 = 8$	$d'_8 = 9$	O	O	O	O	O	O	O	O	O	R	O
$d_9 = 9$	$d'_9 = \infty$	O	O	O	O	O	O	O	O	O	O	O
$next(j)$		2	2	4	4	8	8	8	8	9	∞	∞

Theorem 6. *For $0 \le k \le L$, if $next(S_{0,k}) \neq \infty$ and $next(S_{0,k}) < S_{0,k+1}$, then $S'_{0,k+1} = next(S_{0,k})$; otherwise, $S'_{0,k+1} = S_{0,k+1}$.*

Proof. $S_{0,k+1}$ stores the smallest index j such that $|LCS(A, B_{1..j})| = k+1$. If $next(S_{0,k}) < S_{0,k+1} = j$, then we have $j' = next(S_{0,k}) < j$ and $b_{j'} = \sigma$ such that $|LCS(A\sigma, B_{1..j'})| = k+1$, so $S'_{0,k+1} = next(S_{0,k})$; otherwise, $S'_{0,k+1} = S_{0,k+1}$. □

In the concept of Theorem 6, the numbers in S may be replaced with some smaller numbers to form S'. With Theorem 6, $S'_{0,*}$ can be easily obtained with a linear scan scheme, as an example $S'_{0,*} = \langle 0, 2, 7, 8 \rangle$ shown in Fig. 8. Theorem 6 can also be applied to each row of S. But we do not really establish the whole row of S'. Our goal is to establish the linear-space S-table, $S'_{0,*}$ and D'.

In each row i of S or S', the unique new appearance is d_i or d'_i, respectively. To compute D' in linear time, we use an array Z to record the status of each index j of B. Here, Z, $next(\cdot)$ and d_i are used to compute d'_i and then to update Z, so Z values in iteration i correspond to S'_{i-1}.

Definition 12 *(Z_j).* Z_j records the replacement status of index j of B (i.e., b_j). There are four possible status symbols for Z_j, explained as follows.

- O: The index j is an old number which has appeared in S when calculating d'_i.
- R: The index j, with $b_j = \sigma$, is used to replace another larger index with status O. In other words, a larger index with status O is replaced by this index j with status R.
- N: There is no replacement when the index j is encountered. In other words, we do not need to check the replacement operation.
- U: The status of the index j is uncertain currently. It may become R, N, or O, in the future.

We first use an example, shown in Table 8, to conceptually explain the construction process of D' from D with Z. In Fig. 8, $S_{0,*} = \langle 0, 5, 7 \rangle$ and $n = |B| = 9$. So we set $Z_0 = Z_5 = Z_7 = O$, which means indexes $0, 5, 7$ are old numbers in $S_{0,*}$. Initially, it sets $Z = OUUUUOUOUUO$, where $Z_{n+1} = Z_{10} = O$ is an auxiliary boundary for Z_∞. Now, $b_2 = b_4 = b_8 = b_9 = \sigma = \texttt{g}$. $S'_{0,*} = \langle 0, 2, 7, 8 \rangle$ is constructed with Theorem 6, where $S'_{0,1} = 2$ can replace a larger index $S_{0,1} = 5$, and $S'_{0,3} = 8$ can replace $S_{0,3} = \infty$. Then, we set $Z_2 = Z_8 = R$. Now, $Z = OURUUOUORUO$ temporarily. In addition, if an isolated status U is directly surrounded by O or R, it can be changed to N (the reason explained later). By treating the dash symbol (–) as a single U, $Z_1 = U$ and $Z_6 = U$ are of types $O–R$ and $O–O$, respectively, then Z_1 and Z_6 are changed to N. The others cannot be determined now. So, we have $Z = ONRUUONORUO$.

For $d_1 = 1$, it first appears in $S_{1,*}$. Currently, $Z_1 = N$, meaning that 1 can replace nothing. That is, 1 is not affected by the addition of σ to A. So, 1 also first appears in $S'_{1,*}$. We get $d'_1 = 1$, and set $Z_1 = O$.

For $d_2 = 2$, it first appears in $S_{2,*}$ (see Fig. 8), but 2 is already in $S'_{1,*}$, where $S'_{1,1} = 2$ originally replaces $S_{1,1} = 5$. It seems that 5 should appear in $S'_{2,*}$ for $b_5 = \texttt{a}$. But $next(2) = 4$, smaller than 5, should appear earlier where $b_2 = b_4 = \texttt{g}$. So 4 becomes the new comer. Currently, $Z_2 = R$ means that 2 can replace a larger index. So we get $d'_2 = 4$, and update $Z_2 = O$, $Z_4 = R$, meaning that 4 can replace a larger index in the future. At the same time, 3 is not affected by appending σ into A since 3 is between 2 and 4 (type $O–R$). Thus, $Z_3 = N$ is updated.

For $d_3 = 3$, currently $Z_3 = N$ means that 3 replaces nothing. Thus, we get $d'_3 = 3$, and update $Z_3 = O$.

For $d_4 = 4$, $Z_4 = R$ means it may replace the following index 5 with $Z_5 = O$ (type $R–O$) or update by the index $next(d_4) = 8$. Since $5 < 8$, we get $d'_4 = 5$ and update $Z_4 = O$.

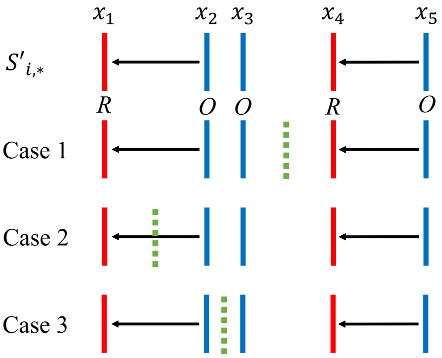


Fig. 9. Three cases for illustrating the situation when d_i appears in different positions. Each vertical represents a number, where the left one is less than the right one. The dotted green line is d_i. The red lines ($Z_j = R$) represent some of the position indexes of the new character σ in B. The arrow means the right one (blue line, $Z_j = O$) is replaced by the left one (red line) in S'.

For $d_5 = 6$, currently $Z_6 = N$. Thus, we get $d'_5 = 6$, and update $Z_6 = O$.

For $d_6 = \infty$ and $d_7 = \infty$, by Theorem 4, we have $d'_6 = \infty$ and $d'_7 = \infty$.

For $d_8 = 8$, $next(8) = 9$ is less than the following index 10 with $Z_{10} = O$. Thus, $d'_8 = 9$, $Z_8 = O$ and $Z_9 = R$, meaning 9 can replace a larger index in the future.

For $d_9 = 9$, it is the last (equal to n) and it can replace ∞. Thus $d'_9 = \infty$ and $Z_9 = O$.

When updating Z step by step, the index j, from status $Z_j = U$ to $Z_j = R$, is obtained by $j = next(i)$ for some $Z_i = O$, where $b_j = \sigma$. According to the left $Z_i = O$, there is at most one $Z_j = Z_{next(i)} = R$ between two consecutive O–O. After processing Z_j, we update $Z_j = O$ and find the next index to set $Z_{next(j)}$ to R, where $b_{next(j)} = \sigma$. This $Z_{next(j)}$ is either a new R closer to $Z_j = O$ or an existing R closer to another O. There is at least one O between two consecutive R–R, so there is no R–R type.

To maintain array Z, Fig. 9 shows three cases of different positions of d_i. The blue lines represent the old numbers ($Z_j = O$) in $S_{i,*}$, and the red lines represent some of the position indexes of the new character σ in B ($Z_j = R$). The blue lines will be replaced by the red lines to form $S'_{i,*}$. In the top case, $S_{i,*} = \langle x_2, x_3, x_5 \rangle$ and $S'_{i,*} = \langle x_1, x_3, x_4 \rangle$. We discuss the three positions that d_i may appear as follows.

1. Case 1: Type O–R, $d_i \in (x_3, x_4)$. x_4 replaces x_5. No number in (x_3, x_4) can replace d_i since $next(x_3) = x_4 \geq d_i$. Therefore, $d'_i = d_i$. The status in this range is N, because no replacement is done. d_1 and d_3 in Table 8 fall in this case.
2. Case 2: Type R–O, $d_i \in [x_1, x_2)$. Note that d_i may be a matching position for new σ, thus, d_i may be equal to x_1. x_1 becomes the replacer of d_i. We need to find a new replacer for x_2. If $next(d_i) < x_2$, then $next(d_i)$ becomes the new replacer number of x_2. Therefore, $d'_i = next(d_i)$, and $Z_{next(d_i)} = R$. If $next(d_i) \geq x_2$, no number can replace x_2. Thus, $d'_i = x_2$. This case also happens if $d_i \in [x_4, x_5)$. The original status in this range is U, we need to update the replacement status for different d_i. d_2, d_4, d_8 and d_9 of Table 8 fall in this case.
3. Case 3: Type O–O, $d_i \in (x_2, x_3)$. Clearly, no number can replace d_i. Therefore, $d'_i = d_i$. The status in this range is N. d_5 of Table 8 falls in this case.

In summary, we first compute $S'_{0,*}$, and set the array Z as follows.

- If $j \in S_{0,*}$, set $Z_j = O$.
- If $j \in S'_{0,*} \setminus S_{0,*}$, set $Z_j = R$.
- If j is of type O–R or O–O, set $Z_j = N$.
- If j is none of the above, set $Z_j = U$.

Then, the operations of the subsequent steps for computing each d'_i and updating Z are given as follows.

- If $Z_{d_i} = N$, set $d'_i = d_i$ and $Z_{d_i} = O$.
- If $Z_{d_i} = R$ or U, find the smallest index j, $j \geq d_i + 1$, such that $next(d_i) = j$ or $Z_j = O$. Set $d'_i = j$. The status Z is updated: $Z_{d_i} = O$, $Z_{d_i+1..j-1} = N$ (original status is U), and if $next(d_i) = j$, set $Z_j = R$.

The algorithm is formally presented in Algorithm 4. We can calculate $S'_{0,*}$ and initial Z with a linear scan scheme, in $O(n)$ time. To compute d'_i, we analyze the status modified in Z. When we encounter a number with status N, we change N to O. When we encounter a number with status R, we change R to O. When we encounter a number with status U, we change U to N, R or O. After the initialization of Z, we only access and modify twice for each number in Z. Thus, the total time complexity is $O(n)$. For the working space, we need an array Z to maintain the status, so the space complexity is $O(n)$.

Algorithm 4 The computation of the linear-space S-table with a new element σ appended in the tail of A.

```
Input: $S_{0,*}$ and $D$ of $A$ and $B$
Output: $S'_{0,*}$ and $D'$ of $A\sigma$ and $B$
compute $next$ and $S'_{0,*}$ and initialize $Z$
for $i = 1$ to $n$ do
    if $d_i = \infty$ then
        $d'_i \leftarrow \infty$                                        ▷ Theorem 4
    else if $Z_{d_i} = N$ then
        update $Z_{d_i} \leftarrow O$ and $d'_i \leftarrow d_i$
    else
        update $Z_{d_i} \leftarrow O$
        if $i = n$ then
            $d'_i \leftarrow \infty$
        else
            for $j = d_i + 1$ to $n$ do
                if $Z_j = O$ then
                    $d'_i \leftarrow j$
                    break
                else if $j \leftarrow next(d_i)$ then
                    update $Z_j \leftarrow R$ and $d'_i \leftarrow j$
                    break
                update $Z_j \leftarrow N$          ▷ set to $N$ between $d_i + 1$ and next $O$ or $next(d_i)$
```

7. Conclusion

The longest common subsequence (LCS) method is a fundamental technique for estimating the similarity between sequences. The S-table can be used to solve the consecutive suffix alignment problem and some LCS related problems. In the previous studies of the S-table, the whole S-table of quadratic space is required for further applications. Due to the growth of data size, reducing the computational space is an important issue. This paper considers the linear-space S-table, which consists of the first row of the S-table and the changes between every two consecutive rows.

New algorithms are proposed to solve the concatenated LCS problem in $O(n \log n)$ and $O(n)$ time when given the linear-space S-table, instead of the whole S-table reconstruction. Then, we propose a new algorithm to merge two linear-space S-tables in $O(n \log n)$ time to improve the $O(nL)$ time of the straightforward method on the quadratic-space S-tables. Finally, an $O(n)$-time algorithm is presented to compute the linear-space S-table of $A\sigma$ and B when given the linear-space S-table of A and B, where σ denotes a new character appended to the tail of A.

Crochemore et al. [27] proposed a compression method for the LCS problem, which divides a string into several substrings and then tries to concatenate them. By constructing a linear-space S-table for each substring of the input string, our algorithms may be applied to solving the LCS problem of compressed strings. However, the tight lower bound of time complexity for solving the LCS problem with various parameters, such as letter size, sequence similarity and LCS length, has been studied [23–25]. This implies that improving the LCS algorithms with compressed sequences may be a challenging task. It may be interesting to study whether the time complexity for merging two linear-space S-tables can be improved or not in the future.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

References

[1] B.-S. Lin, K.-T. Tseng, C.-B. Yang, K.-S. Huang, Fast algorithms for the concatenated longest common subsequence problem with the linear-space S-table, in: Proc. of the 35th Workshop on Combinatorial Mathematics and Computation Theory, Taichung, Taiwan, 2018, pp. 22–30.
[2] L. Allison, T.I. Dix, A bit-string longest-common-subsequence algorithm, Inf. Process. Lett. 23 (5) (1986) 305–310.
[3] A. Apostolico, String editing and longest common subsequences, in: Handbook of Formal Languages, Springer, 1997, pp. 361–398.
[4] L. Bergroth, H. Hakonen, T. Raita, A survey of longest common subsequence algorithms, in: Proceedings of Seventh International Symposium on String Processing and Information Retrieval, IEEE, A Coruña, Spain, 2000, pp. 39–48.
[5] M. Crochemore, C.S. Iliopoulos, Y.J. Pinzon, J.F. Reid, A fast and practical bit-vector algorithm for the longest common subsequence problem, Inf. Process. Lett. 80 (6) (2001) 279–285.
[6] D.S. Hirschberg, A linear space algorithm for computing maximal common subsequences, Commun. ACM 18 (6) (1975) 341–343.
[7] D.S. Hirschberg, Algorithms for the longest common subsequence problem, J. ACM 24 (4) (1977) 664–675.
[8] J.W. Hunt, T.G. Szymanski, A fast algorithm for computing longest common subsequences, Commun. ACM 20 (5) (1977) 350–353.
[9] M. Lu, H. Lin, Parallel algorithms for the longest common subsequence problem, IEEE Trans. Parallel Distrib. Syst. 5 (8) (1994) 835–848.

[10] S.B. Needleman, C.D. Wunsch, A general method applicable to the search for similarities in the amino acid sequence of two proteins, J. Mol. Biol. 48 (3) (1970) 443–453.
[11] R.A. Wagner, M.J. Fischer, The string-to-string correction problem, J. ACM 21 (1) (1974) 168–173.
[12] S. Grabowski, New tabulation and sparse dynamic programming based techniques for sequence similarity problems, Discrete Appl. Math. 212 (2016) 96–103.
[13] K.-S. Huang, C.-B. Yang, K.-T. Tseng, H.-Y. Ann, Y.-H. Peng, Efficient algorithms for finding interleaving relationship between sequences, Inf. Process. Lett. 105 (5) (2008) 188–193.
[14] Y.-H. Peng, C.-B. Yang, K.-S. Huang, C.-T. Tseng, C.-Y. Hor, Efficient sparse dynamic programming for the merged LCS problem with block constraints, Int. J. Innov. Comput. Inf. Control 6 (4) (2010) 1935–1947.
[15] A.H.M.M. Rahman, M.S. Rahman, Effective sparse dynamic programming algorithms for merged and block merged LCS problems, J. Comput. 9 (8) (2014) 1743–1754.
[16] K.-T. Tseng, D.-S. Chan, C.-B. Yang, S.-F. Lo, Efficient merged longest common subsequence algorithms for similar sequences, Theor. Comput. Sci. 708 (2018) 75–90.
[17] H.-Y. Ann, C.-B. Yang, C.-T. Tseng, Efficient polynomial-time algorithms for the constrained LCS problem with strings exclusion, J. Comb. Optim. 28 (4) (2014) 800–813.
[18] H.-Y. Ann, C.-B. Yang, C.-T. Tseng, C.-Y. Hor, Fast algorithms for computing the constrained LCS of run-length encoded strings, Theor. Comput. Sci. 432 (2012) 1–9.
[19] F.Y. Chin, A. De Santis, A.L. Ferrara, N. Ho, S. Kim, A simple algorithm for the constrained sequence problems, Inf. Process. Lett. 90 (4) (2004) 175–179.
[20] Y.-H. Peng, C.-B. Yang, K.-T. Tseng, K.-S. Huang, An algorithm and applications to sequence alignment with weighted constraints, Int. J. Found. Comput. Sci. 21 (1) (2010) 51–59.
[21] Y.-T. Tsai, The constrained longest common subsequence problem, Inf. Process. Lett. 88 (4) (2003) 173–176.
[22] C.-T. Tseng, C.-B. Yang, H.-Y. Ann, Efficient algorithms for the longest common subsequence problem with sequential substring constraints, J. Complex. 29 (1) (2013) 44–52.
[23] A. Abboud, A. Backurs, V.V. Williams, Tight hardness results for LCS and other sequence similarity measures, in: 2015 IEEE 56th Annual Symposium on Foundations of Computer Science, USA, 2015, pp. 59–78.
[24] K. Bringmann, M. Künnemann, Quadratic conditional lower bounds for string problems and dynamic time warping, in: 2015 IEEE 56th Annual Symposium on Foundations of Computer Science, USA, 2015, pp. 79–97.
[25] K. Bringmann, M. Künnemann, Multivariate fine-grained complexity of longest common subsequence, in: Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, USA, 2018, pp. 1216–1235.
[26] G.M. Landau, E. Myers, M. Ziv-Ukelson, Two algorithms for LCS consecutive suffix alignment, J. Comput. Syst. Sci. 73 (7) (2007) 1095–1117.
[27] M. Crochemore, G.M. Landau, M. Ziv-Ukelson, A subquadratic sequence alignment algorithm for unrestricted scoring matrices, SIAM J. Comput. 32 (6) (2003) 1654–1673.
[28] G.M. Landau, M. Ziv-Ukelson, On the common substring alignment problem, J. Algorithms 41 (2) (2001) 338–359.
[29] G.M. Landau, B. Schieber, M. Ziv-Ukelson, Sparse LCS common substring alignment, Inf. Process. Lett. 88 (6) (2003) 259–270.
[30] G.M. Landau, E.W. Myers, J.P. Schmidt, Incremental string comparison, SIAM J. Comput. 27 (2) (1998) 557–582.
[31] M. Maes, On a cyclic string-to-string correction problem, Inf. Process. Lett. 35 (2) (1990) 73–78.
[32] J.P. Schmidt, All highest scoring paths in weighted grid graphs and their application to finding all approximate repeats in strings, SIAM J. Comput. 27 (4) (1998) 972–992.
[33] C.E.R. Alves, E.N. Cáceres, S.W. Song, An all-substrings common subsequence algorithm, Electron. Notes Discrete Math. 19 (2005) 133–139.
[34] A. Aggarwal, M.M. Klawe, S. Moran, P. Shor, R. Wilber, Geometric applications of a matrix searching algorithm, Algorithmica 2 (1987) 195–208.
[35] J.L. Bentley, Algorithms for Klee's rectangle problems, Technical Report, Computer Science Department, Carnegie Mellon University, 1977.
[36] D. Eppstein, S. Muthukrishnan, Internet packet filter management and rectangle geometry, in: Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '01, USA, 2001, pp. 827–835.
[37] H.N. Gabow, R.E. Tarjan, A linear-time algorithm for a special case of disjoint set union, J. Comput. Syst. Sci. 30 (2) (1985) 209–221.
[38] R.E. Tarjan, Efficiency of a good but not linear set union algorithm, J. ACM 22 (2) (1975) 215–225.