Information Processing Letters 129 (2018) 11–15

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

A hardness result and new algorithm for the longest common palindromic subsequence problem

Shunsuke Inenaga [a,*], Heikki Hyyrö [b]

[a] Department of Informatics, Kyushu University, Japan
[b] Faculty of Natural Sciences, University of Tampere, Finland

ARTICLE INFO

Article history:
Received 16 January 2017
Accepted 29 August 2017
Available online 11 September 2017
Communicated by R. Uehara

Keywords:
Algorithms
String processing
Palindromic subsequences
Longest common subsequences
Nesting rectangles

ABSTRACT

The 2-*LCPS problem*, first introduced by Chowdhury et al. (2014) [17], asks one to compute (the length of) a longest common palindromic subsequence between two given strings A and B. We show that the 2-LCPS problem is at least as hard as the well-studied longest common subsequence problem for four strings. Then, we present a new algorithm which solves the 2-LCPS problem in $O(\sigma M^2 + n)$ time, where n denotes the length of A and B, M denotes the number of matching positions between A and B, and σ denotes the number of distinct characters occurring in both A and B. Our new algorithm is faster than Chowdhury et al.'s sparse algorithm when $\sigma = o(\log^2 n \log\log n)$.

1. Introduction

Given $k \geq 2$ string, the *longest common subsequence problem* for k strings (*k-LCS problem* for short) asks to compute (the length of) a longest string that appears as a subsequence in all the k strings. Whilst the problem is known to be NP-hard for arbitrary many strings [1], it can be solved in polynomial time for a constant number of strings (namely, when k is constant).

The 2-LCS problem that concerns two strings is the most basic, but also the most widely studied and used, form of longest common subsequence computation. Indeed, the 2-LCS problem and similar two-string variants are central topics in theoretical computer science and have applications e.g. in computational biology, spelling correction, optical character recognition and file versioning. The fundamental solution to the 2-LCS problem is based on dynamic programming [2] and takes $O(n^2)$ for two given strings of length n.[1] Using the so-called "Four Russians" technique [3], one can solve the 2-LCS problem for strings over a constant alphabet in $O(n^2/\log^2 n)$ time [4]. For a non-constant alphabet, the 2-LCS problem can be solved in $O(n^2 \log\log n/\log^2 n)$ time [5]. Despite much effort, these have remained as the best known algorithms to the 2-LCS problem, and no strongly sub-quadratic time 2-LCS algorithm is known. Moreover, the following conditional lower bound for the 2-LCS problem has been shown: For any constant $\lambda > 0$, an $O(n^{2-\lambda})$-time algorithm which solves the 2-LCS problem over an alphabet of size 7 refutes the so-called strong exponential time hypothesis (SETH) [6].

In many applications it is reasonable to incorporate additional constraints to the LCS problem (see e.g. [7–16]). Along this line of research, Chowdhury et al. [17] introduced the *longest common palindromic subsequence problem* for two strings (2-*LCPS problem* for short), which asks one to compute (the length of) a longest common subsequence

* Corresponding author.
E-mail addresses: inenaga@inf.kyushu-u.ac.jp (S. Inenaga), heikki.hyyro@uta.fi (H. Hyyrö).

[1] For simplicity, we assume that input strings are of equal length n. However, all algorithms mentioned and proposed in this paper are applicable for strings of different lengths.

http://dx.doi.org/10.1016/j.ipl.2017.08.006
0020-0190/

between strings A and B with the additional constraint that the subsequence must be a palindrome. The problem is equivalent to finding (the length of) a longest palindrome that appears as a subsequence in both strings A and B, and is motivated for biological sequence comparison [17]. Chowdhury et al. presented two algorithms for solving the 2-LCPS problem. The first is a conventional dynamic programming algorithm that runs in $O(n^4)$ time and space. The second uses sparse dynamic programming and runs in $O(M^2 \log^2 n \log\log n + n)$ time and $O(M^2)$ space,[2] where M is the number of matching position pairs between A and B.

The contribution of this paper is two-folds: Firstly, we show a tight connection between the 2-LCPS problem and the 4-LCS problem by giving a simple linear-time reduction from the 4-LCS problem to the 2-LCPS problem. This means that the 2-LCPS problem is at least as hard as the 4-LCS problem, and thus achieving a significant improvement on the 2-LCPS problem implies a breakthrough on the well-studied 4-LCS problem, to which all existing solutions [18–22] require at least $O(n^4)$ time in the worst case. Secondly, we propose a new algorithm for the 2-LCPS problem which runs in $O(\sigma M^2 + n)$ time and uses $O(M^2 + n)$ space, where σ denotes the number of distinct characters occurring in both A and B. We remark that our new algorithm is faster than Chowdhury et al.'s sparse algorithm with $O(M^2 \log^2 n \log\log n + n)$ running time [17] when $\sigma = o(\log^2 n \log\log n)$.

2. Preliminaries

Let Σ be an *alphabet*. An element of Σ is called a *character* and that of Σ^* is called a *string*. For any string $A = a_1 a_2 \cdots a_n$ of length n, $|A|$ denotes its length, that is, $|A| = n$.

For any string $A = a_1 \cdots a_m$, let A^R denote the reverse string of A, namely, $A^R = a_m \cdots a_1$. A string P is said to be a *palindrome* iff P reads the same forward and backward, namely, $P = P^R$.

A string S is said to be a *subsequence* of another string A iff there exist increasing positions $1 \le i_1 < \cdots < i_{|S|} \le |A|$ in A such that $S = a_{i_1} \cdots a_{i_{|S|}}$. In other words, S is a subsequence of A iff S can be obtained by removing zero or more characters from A.

A string S is said to be a *common subsequence* of k strings ($k \ge 2$) iff S is a subsequence of all the k strings. S is said to be a *longest common subsequence* (*LCS*) of the k strings iff other common subsequences of the k strings are not longer than S. The problem of computing (the length of) an LCS of k strings is called the *k-LCS problem*.

A string P is said to be a *common palindromic subsequence* of k strings ($k \ge 2$) iff P is a palindrome and is a subsequence of all these k strings. P is said to be a *longest common palindromic subsequence* (*LCPS*) of the k strings iff other common palindromic subsequences of the k strings are not longer than P.

In this paper, we consider the following problem:

Problem 1 *(The 2-LCPS problem).* Given two strings A and B, compute (the length of) an LCPS of A and B.

For two strings $A = a_1 \cdots a_n$ and $B = b_1 \cdots b_n$, an ordered pair (i, j) with $1 \le i, j \le n$ is said to be a *matching position pair* between A and B iff $a_i = b_j$. Let M be the number of matching position pairs between A and B. We can compute all the matching position pairs in $O(n + M)$ time for strings A and B over integer alphabets of polynomial size in n.

3. Reduction from 4-LCS to 2-LCPS

In this section, we show that the 2-LCPS problem is at least as hard as the 4-LCS problem.

Theorem 1. *The 4-LCS problem can be reduced to the 2-LCPS problem in linear time.*

Proof. Let A, B, C, and D be four input strings for the 4-LCS problem. We wish to compute an LCS of all these four strings. For simplicity, assume $|A| = |B| = |C| = |D| = n$. We construct two strings $X = A^R Z B$ and $Y = C^R Z D$ of length $4n+1$ each, where $Z = \$^{2n+1}$ and \$ is a single character which does not appear in A, B, C, or D. Then, since Z is a common palindromic subsequence of X and Y, and since $|Z| = 2n+1$ while $|A|+|B| = |C|+|D| = 2n$, any LCPS of X and Y must be at least $2n+1$ long containing Z as a substring. This implies that the alignment for any LCPS of X and Y is enforced so that the two Z's in X and Y are fully aligned. Since any LCPS of X and Y is a palindrome, it must be of form $T^R Z T$, where T is an LCS of A, B, C, and D. Thus, we can solve the 4-LCS problem by solving the 2-LCPS problem. □

Example 1. Consider four strings A = aabbccc, B = aabbcaa, C = aaabccc, and D = abcbbbb of length 7 each. Then, an LCPS of X = cccbbaa$\$^{15}$aabbcaa and Y = cccbaaa$\$^{15}$abcbbbb is cba$\$^{15}$abc, which is obtained by e.g., the following alignment:

```
c c c b b a a $ $ $ $ $ $ $ $ $ $ $ $ $ $ $ a a b b c a a
    | /     | | | | | | | | | | | | | | | | |  /  /
c c c b a a a $ $ $ $ $ $ $ $ $ $ $ $ $ $ $ a b c b b b b
```

Observe that abc is an LCS of A, B, C, and D.

4. A new algorithm for 2-LCPS

In this section, we present a new algorithm for the 2-LCPS problem.

4.1. Finding rectangles with maximum nesting depth

Our algorithm follows the approach used in the sparse dynamic programming algorithm by Chowdhury et al. [17]:

[2] The original time bound claimed in [17] is $O(M^2 \log^2 n \log\log n)$, since they assume that the matching position pairs are already computed. For given strings A and B of length n each over an integer alphabet of polynomial size in n, we can compute all matching position pairs of A and B in $O(M + n)$ time.