Journal of Mathematics and Informatics
Vol. 27, 2024, 1-7
ISSN: 2349-0632 (P), 2349-0640 (online)
Published 12 August 2024
www.researchmathsci.org
DOI:http://dx.doi.org/10.22457/jmi.v27a01245

Journal of
Mathematics and Informatics

An Algorithm for the Constrained Longest Common Substring Problem

Rao Li [*1] **and** ***Richy Modugu*** [2]

[1]Department of Computer Science, Engineering, and Mathematics,
University of South Carolina Aiken, Aiken, SC 29801, USA,
E-mail: raol@usca.edu
[2]Dept. of Computer Science, Engineering, and Mathematics,
University of South Carolina Aiken, Aiken, SC 29801, USA,
E-mail: rmodugu@usca.edu
[*]Corresponding author

Received 11 July 2024; accepted 12 August 2024

Abstract. Let ∑ be an alphabet. For two strings X, Y, and a constrained string P over the alphabet ∑, the constrained longest common substring problem for two strings X and Y with respect to P is to find a longest string Z which is a substring of both X and Y and has P as a subsequence. In this paper, we propose an algorithm for the constrained longest common substring problem for two strings with a constrained string.

Keywords: The longest common substring, the constrained longest common substring

AMS Mathematics Subject Classification (2010): 68W32, 68W40

1. Introduction

Let ∑ be an alphabet and S a string over ∑. The size of S, denoted |S|, is defined as the number of letters in S. A subsequence of a string S over an alphabet ∑ is obtained by deleting zero or more letters of S. A substring of a string S is a subsequence of S consists of consecutive letters in S. The longest common subsequence (LCSSeq) problem for two strings is to find a longest string which is a subsequence of both strings. The longest common substring (LCSStr) problem for two strings is to find a longest string which is a substring of both strings. Both the longest common subsequence problem and the longest common substring problem have been well-studied in the last several decades. More details on the studies for the first problem can be found in [1-4,6,9-11,14-15] and the second problem can be found in [7-8,17].

Tsai [16] extended LCSSeq problem to the constrained longest common subsequence (CLCSSeq) problem. For two strings X, Y, and a constrained string P, the constrained longest common subsequence problem for X and Y with respect to P is to find a string Z such that Z is a longest common subsequence for both X and Y and P is a subsequence of Z. Clearly, the LCSSeq problem is a special CLCSSeq problem with an empty constrained string.

Motivated by Tsai's extension of LCSSeq problem to CLCSSeq problem, we introduce the constrained longest common substring problem for two strings and a constrained string in this paper. For two strings X and Y and a constrained string P, the constrained longest common substring (CLCSStr) problem for X and Y with respect to P is to find a string Z such that Z is a longest common string of both X and Y and has P as a subsequence. Clearly, the LCSStr problem is a special CLCSStr problem with an empty constrained string. The other related problems and the research on them can be found in [12] and [13]. In this paper, we, using some ideas in [5], design an O(|X| |Y| |P|) time algorithm for CLCSSStr problem for two strings X and Y and a constrained string P.

2. The recursions in the algorithm

In order to present our algorithm, we need to establish some recursions to be used in our algorithm. Before establishing the recursions, we need some notations as follows. For a given string $S = s_1s_2 \ldots s_l$ over an alphabet $\sum$. The ith prefix of S is defined as $S_i = s_1s_2 \ldots s_i$, where $1 \leq i \leq l$. Conventionally, S_0 is defined as an empty string. The l suffixes of S are the strings of $s_1s_2 \ldots s_l$, $s_2s_3 \ldots s_l$, ..., $s_{l-1}s_l$, and s_l. Let $X = x_1x_2 \ldots x_m$ and $Y = y_1y_2 \ldots y_n$ be two strings and $P = p_1p_2 \ldots p_r$ a constrained string. We define Z[i, j, k] as a string satisfying the following conditions, where $1 \leq i \leq m$, $1 \leq j \leq n$, and $1 \leq k \leq r$,

(1) it is a suffix of X_i,
(2) it is a suffix of Y_j,
(3) it has P_k as a subsequence,
(4) under (1), (2) and (3), its length is as large as possible.

Claim 1. Let $U^k = u_1^k u_2^k \ldots u_{h\_k}^k\$$ be a longest string which is a substring of both X and Y and has P_k as a subsequence. Then h_k = max{|Z[i, j, k]| : $1 \leq i \leq m$, $1 \leq j \leq n$}, where k is fixed such that $0 \leq k \leq r$.

Proof of Claim 1. For each i with $1 \leq i \leq m$, each j with $1 \leq j \leq n$, and a fixed k with $0 \leq k \leq r$, we, from the definition of Z[i, j, k], have that Z[i, j, k] is a substring of both X and Y and has P_k as a subsequence. By the definition of U^k, we have that $|Z[i, j, k]| \leq |U^k|$ = h_k. Thus max{|Z[i, j, k]| : $1 \leq i \leq m$, $1 \leq j \leq n$, $1 \leq k \leq r$ } $\leq$ h_k, where k is fixed such that $0 \leq k \leq r$.

For a fixed k with $0 \leq k \leq r$, since $U^k = u_1^k u_2^k \ldots u_{h\_k}^k$ is a (longest) string which is a substring of both X and Y and has P_k as a subsequence, there is an index s, where $1 \leq s \leq m$, and an index t, where $1 \leq t \leq n$, such that $u_{h\_k}^k = x_s$ and $u_{h\_k}^k = y_t$ such that $U^k = u_1^k u_2^k \ldots u_{h\_k}^k$ is a suffix of both X_s and Y_t and has P_k as a subsequence. From the definition of Z[i, j, k], we have that h_k $\leq$ |Z[s, t, k]| $\leq$ max{|Z[i, j, k]| : $1 \leq i \leq m$, $1 \leq j \leq n$}, where k is fixed such that $0 \leq k \leq r$.

Hence h_k = max{|Z[i, j, k]| : $1 \leq i \leq m$, $1 \leq j \leq n$ }, where k is fixed such that $0 \leq k \leq r$, and the proof of Claim 1 is completed.

Claim 2. Suppose that $X_i = x_1x_2 \ldots x_i$, $Y_j = y_1y_2 \ldots y_j$, and $P = p_1p_2 \ldots p_k$, where $1 \leq i \leq m$, $1 \leq j \leq n$, and $1 \leq k \leq r$. If $Z[i, j, k] = z_1z_2 \ldots z_a$ is a string satisfying conditions (1), (2), (3), and (4) above. Then we have only the following possible cases and the statement in each case is true.

Case 1. $x_i = y_j = p_k$. We have that $|Z[i, j, k]| = |Z[i - 1, j - 1, k - 1]| + 1$ in this case.

Case 2. $x_i = y_j \neq p_k$. We have that $|Z[i, j, k]| = |Z[i - 1, j - 1, k]| + 1$ in this case.

Case 3. $x_i \neq y_j$, $x_i \neq p_k$, and $y_j = p_k$. This case cannot happen.

Case 4. $x_i \neq y_j$, $x_i \neq p_k$, and $y_j \neq p_k$. This case cannot happen.

Case 5. $x_i \neq y_j$, $x_i = p_k$, and $y_j \neq p_k$. This case cannot happen.

Proof of Claim 2. The five cases can be figured out in the following way. Firstly, we have two cases of $x_i = y_j$ or $x_i \neq y_j$. When $x_i = y_j$, we just can have two possible subcases of $x_i = y_j = p_k$ or $x_i = y_j \neq p_k$. When $x_i \neq y_j$, we just can have three possible subcases of $x_i \neq p_k$ and $y_j = p_k$, $x_i \neq p_k$ and $y_j \neq p_k$, or $x_i = p_k$ and $y_j \neq p_k$. Next, we will prove the statements in the five cases.

Case 1. Since $Z[i, j, k] = z_1z_2 \ldots z_a$ is a suffix of both X_i and Y_j, we have that $z_a = y_j = x_i = p_k$. Let $W = w_1w_2 \ldots w_b = Z[i - 1, j - 1, k - 1]$ be a string satisfying the following conditions,

- it is a suffix of X_{i-1},
- it is a suffix of Y_{j-1},
- it has P_{k-1} as a subsequence,
- under three conditions above, its length is as large as possible.

Note that $z_1z_2 \ldots z_{a-1}$ is a string which is a suffix of both X_{i-1} and Y_{j-1} and has P_{k-1} as a subsequence. By the definition of $W = w_1w_2 \ldots w_b$, we have that $a - 1 \leq b$. Namely, $a \leq b + 1$.

Note that $w_1w_2 \ldots w_bz_a$ is a string satisfying following conditions,

- it is a suffix of X_i,
- it is a suffix of Y_j,
- it has P_k as a subsequence.

By the definition of $Z[i, j, k] = z_1z_2 \ldots z_a$, we have that $b + 1 \leq a$. Thus $a = b + 1$ and $|Z[i, j, k]| = |Z[i - 1, j - 1, k - 1]| + 1$.

Case 2. Since $Z[i, j, k] = z_1z_2 \ldots z_a$ is a suffix of both X_i and Y_j, we have that $z_a = y_j = x_i \neq p_k$. Let $U = u_1u_2 \ldots u_c = Z[i - 1, j - 1, k]$ be a string satisfying the following conditions,

- it is a suffix of X_{i-1},
- it is a suffix of Y_{j-1},
- it has P_k as a subsequence,
- under three conditions above, its length is as large as possible.

Note that $z_1z_2 \ldots z_{a-1}$ is a string which is a suffix of both X_{i-1} and Y_{j-1} and has P_k as a subsequence. By the definition of $U = u_1u_2 \ldots u_c = Z[i - 1, j - 1, k]$, we have that $a - 1 \leq c$. Namely, $a \leq c + 1$.

Note that $u_1u_2 \ldots u_c$ is a string satisfying the following conditions,

- it is a suffix of X_{i-1},
- it is a suffix of Y_{j-1},
- it has P_k as a subsequence.

Thus $u_1u_2 \dots u_cy_j$ is a string which is a suffix of both X_i and Y_j and has P_k as a subsequence. By the definition of $Z[i, j, k] = z_1z_2 \dots z_a$, we have that $c + 1 \leq a$. Thus $a = c + 1$ and $|Z[i, j, k]| = |Z[i - 1, j - 1, k]| + 1$.

Case 3. By the definition of Z[i, j, k], we have $z_a = x_i$ and $z_a = y_j$. So, this case cannot happen since $x_i \neq y_j$.

Case 4. By the definition of Z[i, j, k], we have $z_a = x_i$ and $z_a = y_j$. So, this case cannot happen since $x_i \neq y_j$.

Case 5. By the definition of Z[i, j, k], we have $z_a = x_i$ and $z_a = y_j$. So, this case cannot happen since $x_i \neq y_j$.

Therefore, the proof of Claim 2 is completed.

The following Claim 3 which will be used in our algorithm demonstrates the implications of the conditions that there is not a string which is a suffix of both $X_i = x_1 x_2 \dots x_i$ and $Y_j = y_1 y_2 \dots y_j$ and has $P_k = p_1p_2 \dots pk$ as a subsequence.

Claim 3. Suppose there is not a string which is a suffix of both $X_i = x_1x_2 \dots x_i$ and $Y_j = y_1 y_2 \dots y_j$ and has $Pk = p_1p_2 \dots p_k$ as a subsequence.
[1]. If $x_i = y_j = p_k$, then there is not a string which is a suffix of both $X_{i-1} = x_1x_2 \dots x_{i-1}$ and $Y_{j-1} = y_1y_2 \dots y_{j-1}$ and has $P_{k-1} = p_1p_2 \dots p_{k-1}$ as a subsequence.
[2]. If $x_i = y_j \neq p_k$, then there is not a string which is a suffix of both $X_{i-1} = x_1 x_2 \dots x_{i-1}$ and $Y_{j-1} = y_1 y_2 \dots y_{j-1}$ and has $P_k = p_1 p_2 \dots p_k$ as a subsequence.

Proof of Claim 3. We next will prove the statements in the two cases.
[1]. Now we have that $x_i = y_j = p_k$. Suppose, to the contrary, that there is a string W_1 which is a suffix of both $X_{i-1} = x_1x_2 \dots x_{i-1}$ and $Y_{j-1} = y_1 y_2 \dots y_{j-1}$ and has $P_{k-1} = p_1 p_2 \dots p_{k-1}$ as a subsequence. Then W_1x_i is a string which is a suffix of both $X_i = x_1x_2 \dots x_i$ and $Y_j = y_1y_2 \dots y_j$ and has $P_k = p_1p_2 \dots p_k$ as a subsequence, a contradiction.
[2]. Now we have that $x_i = y_j \neq p_k$. Suppose, to the contrary, that there is a string W_2 which is a suffix of both $X_{i-1} = x_1x_2 \dots x_{i-1}$ and $Y_{j-1} = y_1y_2 \dots y_{j-1}$ and has $P_k = p_1p_2 \dots p_k$ as a subsequence. Then W_2x_i is a string which is a suffix of both $X_i = x_1x_2 \dots x_i$ and $Y_j = y_1y_2 \dots y_j$ and has $P_k = p_1 p_2 \dots p_k$ as a subsequence, a contradiction.

Therefore, the proof of Claim 3 is completed.

3. The algorithm

Now we can present our algorithm. We assume that $X = x_1x_2 \dots x_m$, $Y = y_1y_2 \dots y_n$, and $P = p_1p_2 \dots p_r$. Let M be a three-dimensional array of size $(m + 1)(n + 1)(r + 1)$. It can be thought as a collection of $(r + 1)$ two-dimensional arrays of size $(m + 1)(n + 1)$. The cells M[i][j][k], where $0 \leq i \leq m$, $0 \leq j \leq n$, and $0 \leq k \leq r$, store the lengths of longest strings such that each of them is a suffix of both X_i and Y_j and has P_k as a subsequence.

If either $i < k$ or $j < k$, there is not a string which is a suffix of both X_i and Y_j and has P_k as a subsequence. This situation is represented by setting $M[i][j][k] = -\infty$, where ∞ should be a larger number, for example, 100mnr. Now we can fill in the boundary cells in array M.

Step 1. If $i = 0$ and $k = 0$ or $j = 0$ and $k = 0$, the length of a string which is a suffix of both X_i and Y_j and has P_k as a subsequence is zero. Thus $M[0][j][0] = 0$, where $0 \le j \le n$, and $M[i][0][0] = 0$, where $0 \le i \le m$.

Step 2. If $k = 0$ or P is an empty string. The CLCSStr problem for two strings X and Y and a constrained string P becomes the LCSStr problem for two strings X and Y. The cells of $M[i][j][0]$, where $1 \le i \le m$ and $1 \le j \le n$, can be filled in by the following rules. If $x_i = y_j$, then $M[i][j][0] = M[i - 1][j - 1] + 1$. If $x_i \ne y_j$, then $M[i][j][0] = 0$. The reasons that the rules work here can be found in [18].

Step 3. If $i = 0$ and $k \ge 1$, there is not a string which is a suffix of both X_i and Y_j and has P_k as a subsequence. Thus $M[0][j][k] = -\infty$, where $0 \le j \le n$ and $1 \le k \le r$.

Step 4. If $j = 0$ and $k \ge 1$, there is not a string which is a suffix of both X_i and Y_j and has P_k as a subsequence. Thus $M[i][0][k] = -\infty$, where $0 \le i \le m$ and $1 \le k \le r$.

Next, we will fill in the remaining cells $M[i][j][k]$, where $i \ge 1$, $j \ge 1$, and $k \ge 1$.

Step 5. If $i \ge 1$, $j \ge 1$, $k \ge 1$, and $x_i = y_j = p_k$, then $M[i][j][k] = M[i - 1][j - 1][k - 1] + 1$.

Step 6. If $i \ge 1$, $j \ge 1$, $k \ge 1$, and $x_i = y_j \ne p_k$, then $M[i][j][k] = M[i - 1][j - 1][k] + 1$.

Step 7. For all the other cases, $M[i][j][k] = -\infty$.

Notice that Claim 1 implies that if a longest string which is a suffix of both $X = X_m$ and $Y = Y_n$ and has $P = P_r$ as a subsequence exists then its length is equal to $\max\{ |Z[i, j, r]| : 1 \le i \le m, 1 \le j \le n\} = \max\{M[i][j][r] : 1 \le i \le m, 1 \le j \le n\}$. Hence, a longest string which is a substring of both X and Y and has P as a subsequence can be found in the following way.

Step 8. Define one variable called *maxLength* which eventually represents the length of a longest string which is a substring of both X and Y and has P as a subsequence and its initial value is *0*.

Step 9. Define another variable called *lastIndexOnY* which eventually represents the last index of the desired string which is a substring of Y and its initial value is *n*.

Step 10. Visit all the cells of $M[i][j][r]$, where $0 \le i \le m$ and $0 \le j \le n$, in the last two dimensional array created in the algorithm above by using a loop embedded another loop. During the visitation, if $M[i][j][r] >$ *maxLength*, then update *maxLength* and *lastIndexOnY* as *M[i][j][r]* and *j*, respectively.

Step 11. After finishing the visitation of all the cells of *M[i][j][r]*, where $0 \le i \le m$ and $0 \le j \le n$, we return the substring of Y between *(lastIndexOnY - maxLength)* and *lastIndexOnY*.

The correctness of the above algorithm is ensured by Claim 1, Claim 2, and Claim 3. It is clear that both time complexity and space complexity of the above algorithm are $O((m + 1)(n + 1)(r + 1)) = O(m\ n\ r)$. We implemented our algorithm in Java and the program can be found at

"https://sciences.usca.edu/math/~mathdept/rli/CLCSubStr/CLCSStr.pdf".

4. Conclusion

In this paper, we introduce a new problem called the constrained longest common substring problem for two strings X and Y and a constrained string P. We propose an algorithm with time complexity and space complexity of O(|X||Y||P|) to solve the problem. In future, we will design new algorithms to improve the time and space complexities and find the applications of our algorithm in the real world.

Acknowledgements. The authors would like to thank the referee for his or her suggestions which led to the improvements of the initial manuscript.

The authors also would like to thank the Summer Scholars Institute (2024) at the University of South Carolina Aiken for its support of this research.

Conflicts of interest. The authors declare no conflicts of interest.

Authors' contributions. All authors contributed equally to this work.

REFERENCES

1. A. Apostolico, String editing and longest common subsequences, in: G. Rozenberg and A. Salomaa (Eds.), Linear Modeling: Background and Application, in: *Handbook of Formal Languages*, Vol. 2, Springer-Verlag, Berlin, 1997.
2. A. Apostolico, Chapter 13: General pattern matching, in: M. J. Atallah (Ed.), *Handbook of Algorithms and Theory of Computation,* CRC, Boca Raton, FL, 1998.
3. L. Bergroth, H. Hakonen and T. Raita, A survey of longest common subsequence algorithms, in: SPIRE, A Corua, Spain, 2000.
4. C. Blum, M. Djukanovic, A. Santini, H. Jiang, C. Li, F. Manyà and G. R. Raidl, Solving longest common subsequence problems via a transformation to the maximum clique problem, *Computers and Operations Research*, 125 (2021) 105089.
5. F. Y. L. Chin, A. De Santis, A. L. Ferrara, N. L. Ho and S. K. Kim, A simple algorithm for the constrained sequence problems, *Information Processing Letters,* 90 (2004) 175-179.
6. T. Cormen, C. Leiserson, and R. Rivest, Section 16.3: Longest common subsequence, Introduction to Algorithms, MIT Press, Cambridge, MA, 1990.
7. M. Crochemore, C. S. Iliopoulos, A. Langiu and F. Mignosi, The longest common substring problem. Mathematical Structures in Computer Science, pp 1-19, Cambridge University Press 2015, doi:10.1017/S0960129515000110.
8. D. Gusfield, *II: Suffix Trees and Their Uses*, *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*, Cambridge University Press, 1997.
9. D. Hirschberg, A linear space algorithm for computing maximal common subsequences, *Communications of the ACM*, 18 (1975) 341-343.
10. D. Hirschberg, Serial computations of Levenshtein distances, in: A. Apostolico and Z.Galil (Eds.), *Pattern Matching Algorithms*, Oxford University Press, Oxford, 1997.
11. J. Hunt and T. Szymanski, A fast algorithm for computing longest common subsequences, *Communications of the ACM*, 20 (1977) 350-353.
12. R. Li, J. Deka, and K. Deka, An algorithm for the longest common subsequence and substring problem, *Journal of Mathematics and Informatics*, 25 (2023) 77-81.
13. R. Li, J. Deka, K. Deka, and D. Li, An algorithm for the constrained longest common subsequence and substring problem, *Journal of Mathematics and Informatics*, 26 (2024) 41-48.
14. S. R. Mousavi and F. Tabataba, An improved algorithm for the longest common subsequence problem, *Computers and Operations Research*, 39 (2012) 512-520.

15. C. Rick, New algorithms for the longest common subsequence problem, Research Report No. 85123-CS, University of Bonn, 1994.
16. Y. T. Tsai, The constrained longest common subsequence problem, *Information Processing Letters*, 88 (2003) 173-176.
17. P. Weiner, Linear pattern matching algorithms. In: *14th Annual Symposium on Switching and Automata Theory*, Iowa City, Iowa, USA, October 15–17, 1973, pp. 1–11 (1973).
18. https://www.geeksforgeeks.org/longest-common-substring-dp-29/. Retrieved on August 10, 2024.