# Computing Longest Common Substring/Subsequence of Non-linear Texts

Kouji Shimohira, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda

Department of Informatics, Kyushu University
744 Motooka, Nishi-ku, Fukuoka 819–0395, Japan
{kouji.shimohira,inenaga,bannai,takeda}@inf.kyushu-u.ac.jp

**Abstract.** A non-linear text is a directed graph where each vertex is labeled with a string. In this paper, we introduce the longest common substring/subsequence problems on non-linear texts. Firstly, we present an algorithm to compute the longest common substring of non-linear texts $G_1$ and $G_2$ in $O(|E_1||E_2|)$ time and $O(|V_1||V_2|)$ space, when at least one of $G_1$ and $G_2$ is acyclic. Here, $V_i$ and $E_i$ are the sets of vertices and arcs of input non-linear text $G_i$, respectively, for $1 \leq i \leq 2$. Secondly, we present algorithms to compute the longest common subsequence of $G_1$ and $G_2$ in $O(|E_1||E_2|)$ time and $O(|V_1||V_2|)$ space, when both $G_1$ and $G_2$ are acyclic, and in $O(|E_1||E_2|+|V_1||V_2|\log|\Sigma|)$ time and $O(|V_1||V_2|)$ space if $G_1$ and/or $G_2$ are cyclic, where, $\Sigma$ denotes the alphabet.

## 1 Introduction

We consider *non-linear texts*, which are directed graphs where vertices are labeled by strings. Pattern matching on non-linear texts was first considered in [3], where an $O(N + m|E| + R\log\log m)$ time algorithm for directed acyclic graphs. Here, $m$ is the pattern length, $N$ is the number of vertices, $|E|$ is the number of arcs, and $R$ is the output size. The algorithm was improved in [6], where an $O(n + m|E|)$ time algorithm was shown. Here, $n$ represents the total length of the string labels in the graph. Furthermore, in [1], an $O(n)$ time algorithm was shown for trees. The problem was solved for general directed graphs in [2], where an $O(n + |E|)$ time algorithm was developed. The approximate matching problem for non-linear texts was also considered in [2], where they showed that the problem can be solved in $O(m(n\log m + e))$ time when edit operations are only allowed in the pattern. Here, $e$ denotes the number of arcs in the graph when the graph is converted so that each node is labeled by a single character. They also showed that the problem is NP-complete when edit operations are allowed on the non-linear text. Furthermore, in [5], the algorithm was improved to $O(m(n + e))$.

Note that previous work on pattern matching on non-linear texts assumed a linear pattern. In this paper, we study a more generalized version of the problem, and consider the *longest common substring* and *longest common subsequence* problems between two non-linear texts. Firstly, we present an algorithm to compute the longest common substring of non-linear texts $G_1$ and $G_2$ in $O(|E_1||E_2|)$ time and $O(|V_1||V_2|)$ space, where $V_i$ and $E_i$ are the sets of vertices and arcs of input non-linear text $G_i$, respectively, for $1 \leq i \leq 2$. The algorithm works if one of $G_1$ and $G_2$ is acyclic. Secondly, we present algorithms to compute the longest common subsequence in $O(|E_1||E_2|)$ time and $O(|V_1||V_2|)$ space if both $G_1$ and $G_2$ are acyclic, and in $O(|E_1||E_2| + |V_1||V_2|\log|\Sigma|)$ time and $O(|V_1||V_2|)$ space if $G_1$ and/or $G_2$ are cyclic. Cyclic non-linear texts represent infinitely many and long strings, but our algorithms solve the above problems quite efficiently. Our algorithms are natural

Kouji Shimohira, Shunsuke Inenaga, Hideo Bannai, Masayuki Takeda: Computing Longest Common Substring/Subsequence of Non-linear Texts, pp. 197–208.
Proceedings of PSC 2011, Jan Holub and Jan Žďárek (Eds.), ISBN 978-80-01-04870-2 

extension of classical dynamic programming methods to compute longest common substring/subsequence of linear strings, and hence are easy to understand.

Very recently, an algorithm for determining the longest common subsequence between two *finite* languages was shown in [7]. The algorithm is a modification of the method based on weighted transducers [4], and requires $O(|\Sigma|^2|E_1||E_2|)$ time and space. Compared to this work, our algorithms are faster and also apply to infinite languages.

| problem | text | pattern | time complexity |
|---|---|---|---|
| Substring Matching | acyclic graph | linear | $O(n + m\|E\|)$ [6] |
| | tree | linear | $O(n)$ [1] |
| | graph | linear | $O(n + \|E\|)$ [2] |
| Approximate Matching | graph w/edit operations | linear | NP-complete [2] |
| | graph | linear w/edit operations | $O(m(n + e))$ [5] |
| | text1 | text2 | |
| Longest Common Substring | acyclic graph | acyclic graph | $O(\|E_1\|\|E_2\|)$ (this work) |
| | graph | acyclic graph | $O(\|E_1\|\|E_2\|)$ (this work) |
| Longest Common Subsequence | acyclic graph | acyclic graph | $O(\|\Sigma\|^2\|E_1\|\|E_2\|)$ [7] |
| | acyclic graph | acyclic graph | $O(\|E_1\|\|E_2\|)$ (this work) |
| | graph | graph | $O(\|E_1\|\|E_2\| + \|V_1\|\|V_2\| \log \|\Sigma\|)$ (this work) |

**Table 1.** Algorithms on non-linear text.

# 2 Preliminaries

## 2.1 Notation

Let $\Sigma$ be a finite alphabet, and the elements of $\Sigma^*$ are called *strings*. The *length* of a string $w$ is denoted by $|w|$. The *empty string*, denoted by $\varepsilon$, is a string of length 0, and thus $|\varepsilon| = 0$. Let $\Sigma^+ = \Sigma^* - \{\varepsilon\}$. Strings $x$, $y$, and $z$ are called a *prefix*, *substring*, and *suffix* of string $w = xyz$, respectively. For any string $w$, let *suffix*$(w)$ denote the set of suffixes of $w$. The $i$-th symbol of a string $w$ is denoted by $w[i]$ for $1 \leq i \leq |w|$, and the substring of $w$ that begins at position $i$ and ends at position $j$ is denoted by $w[i..j]$ for $1 \leq i \leq j \leq |w|$. For convenience, let $w[i..j] = \varepsilon$ for $i > j$. The set of substrings of a string $w$ is denoted by *substr*$(w)$. A string $u$ is a *subsequence* of another string $w$ if there exists a sequence of integers $i_1, \ldots, i_k$ with $k \geq 0$ such that $1 \leq i_1 < \cdots < i_k \leq |w|$ and $u = w[i_1] \cdots w[i_k]$.

A *directed graph* is an ordered pair $(V, E)$ of set $V$ of *vertices* and set $E \subseteq V \times V$ of *arcs*. A *path* in a directed graph $G = (V, E)$ is a sequence $v_0, \ldots, v_k$ of vertices such that $(v_{i-1}, v_i) \in E$ for every $i = 1, \ldots, k$. For any vertex $v \in V$, let $P(v)$ denote the set of paths that end at vertex $v$. The set of all paths in $G$ is denoted by $P(G)$, namely, $P(G) = \{P(v) \mid v \in V\}$.

## 2.2 Longest common substring problem

The *longest common substring* problem is, given two strings $x$ and $y$, to compute the length of longest common substrings of them. Although this problem can be solved in $O(|x| + |y|)$ time using the generalized suffix tree of $x$ and $y$, we here mention a dynamic programming based solution. Letting $C_{i,j}$ denote the maximum length of common suffixes of $x[1..i]$ and $y[1..j]$, it suffices to compute the maximum of $C_{i,j}$ over all the pairs $(i, j)$. Since we have

$$C_{i,j} = \begin{cases} 1 + C_{i-1,j-1} & \text{if } i, j > 0 \text{ and } x[i] = y[j]; \\ 0 & \text{otherwise,} \end{cases} \quad (1)$$

the problem can be solved in $O(|x||y|)$ time.

### 2.3 Longest common subsequence problem

The *longest common subsequence* problem is, given two strings $x$ and $y$, to compute the length of longest common subsequences of them. It is well-known that, this problem can be solved in $O(|x||y|)$ time by using the following recurrence:

$$C_{i,j} = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0; \\ 1 + C_{i-1,j-1} & \text{if } i, j > 0 \text{ and } x[i] = y[j]; \\ \max(C_{i-1,j}, C_{i,j-1}) & \text{if } i, j > 0 \text{ and } x[i] \neq y[j], \end{cases} \quad (2)$$

where $C_{i,j}$ is the length of longest common subsequence of $x[1..i]$ and $y[1..j]$.

### 2.4 Non-linear texts

A *non-linear text* is a directed graph with vertices labeled by strings, namely, it is a directed graph $G = (V, E, L)$ where $V$ is the set of vertices, $E$ is the set of arcs, and $L : V \to \Sigma^+$ is a labeling function that maps nodes $v \in V$ to non-empty strings $L(v) \in \Sigma^+$. For a path $p = v_0, \ldots, v_k \in P(G)$, let $L(p)$ denote the string spelled out by $p$, namely $L(p) = L(v_0) \cdots L(v_k)$. The size $|G|$ of a non-linear text $G = (V, E, L)$ is $|V| + |E| + \sum_{v \in V} |L(v)|$. Let *substr*$(G)$, *suffix*$(G)$, and *subseq*$(G)$ be the sets of substrings, suffices and subsequences of a non-linear text $G = (V, E, L)$, namely,

$$\begin{aligned} \mathit{substr}(G) &= \{\mathit{substr}(L(p)) \mid p \in P(G)\}, \\ \mathit{suffix}(G) &= \{\mathit{suffix}(L(p)) \mid p \in P(G)\}, \\ \mathit{subseq}(G) &= \{\mathit{subseq}(L(p)) \mid p \in P(G)\}. \end{aligned}$$

For a non-linear text $G = (V, E, L)$, consider a non-linear text $G' = (V', E', L')$ such that $L' : V' \to \Sigma$,

$$\begin{aligned} V' &= \{v_{i,j} \mid L'(v_{i,j}) = L(v_i)[j], v_i \in V, 1 \le j \le |L(v_i)|\}, \text{ and} \\ E' &= \{(v_{i,|L(v_i)|}, v_{k,1}) \mid (v_i, v_k) \in E\} \cup \{(v_{i,j}, v_{i,j+1}) \mid v_i \in V, 1 \le j < |L(v_i)|\}. \end{aligned}$$

Namely, $G'$ is a non-linear text in which each vertex is labeled with a single character and *substr*$(G')$ = *substr*$(G)$. An example is shown in Figure 1. Since $|V'| = \sum_{v \in V} |L(v)|$, $|E'| = |E| + \sum_{v \in V}(|L(v)| - 1)$, and $\sum_{v' \in V'} |L(v')| = \sum_{v \in V} |L(v)|$, we have $|G'| = O(|G|)$. We remark that given $G$, we can easily construct $G'$ in $O(|G|)$ time. Observe that *subseq*$(G)$ = *subseq*$(G')$ also holds.

In the sequel we only consider non-linear texts where each vertex is labeled with a single character. For any non-linear text $G = (V, E, L)$ such that $L(v) \in \Sigma$ for any $v \in V$, it trivially holds that *substr*$(G) = \{L(p) \mid p \in P(G)\}$.

We sometimes call strings in $\Sigma^*$ linear strings or linear texts, in order to clearly distinguish them from non-linear texts.

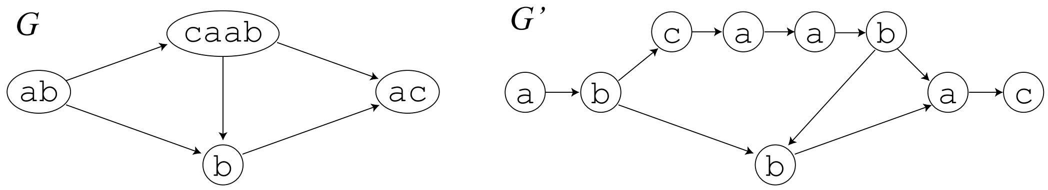



**Figure 1.** A non-linear text $G = (V, E, L)$ with $L : V \to \Sigma^+$ and its corresponding non-linear text $G' = (V', E', L')$ with $L' : V' \to \Sigma$.

# 3 Computing Longest Common Substring of Non-linear Texts

In this section, we tackle the problem of computing the length of longest common substrings of two input non-linear texts. The problem is formalized as follows.

*Problem 1 (Longest common substring problem for non-linear texts).*

**Input:** Non-linear texts $G_1 = (V_1, E_1, L_1)$ and $G_2 = (V_2, E_2, L_2)$.
**Output:** The length of a longest string in $\mathit{substr}(G_1) \cap \mathit{substr}(G_2)$.

For example, see the non-linear texts $G_1$ and $G_2$ of Figure 2. The solution to the above problem is 5, since there is a longest common substring `abbaa` of $G_1$ and $G_2$.

For simplicity, let us first consider the case where the two input non-linear texts are both acyclic.

**Theorem 2.** *If $G_1$ and $G_2$ are acyclic, then Problem 1 can be solved in $O(|E_1||E_2|)$ time and $O(|V_1||V_2|)$ space.*

*Proof.* Let $v_{1,i}$ and $v_{2,j}$ denote the $i$-th and $j$-th vertex in topological ordering in $G_1$ and in $G_2$, for $1 \le i \le |V_1|$ and $1 \le j \le |V_2|$, respectively. Let $C_{i,j}$ denote the length of a longest string in $\mathit{suffix}(L_1(P(v_{1,i}))) \cap \mathit{suffix}(L_2(P(v_{2,j})))$. $C_{i,j}$ can be calculated as follows.

1. If $L_1(v_{1,i}) = L_2(v_{2,j})$, there are two cases to consider:
   (a) If there are no arcs to $v_{1,i}$ or to $v_{2,j}$, i.e., $P(v_{1,j}) = \{v_{1,i}\}$ or $P(v_{2,j}) = \{v_{2,j}\}$, then clearly $C_{i,j} = 1$.
   (b) Otherwise, let $v_{1,k}$ and $v_{2,\ell}$ be any nodes s.t. $(v_{1,k}, v_{1,i}) \in E_1$ and $(v_{2,\ell}, v_{2,j}) \in E_2$, respectively. Let $z$ be a longest string in $\mathit{suffix}(L_1(P(v_{1,i}))) \cap \mathit{suffix}(L_2(P(v_{2,j})))$. Assume on the contrary that there exists a string $y \in \mathit{suffix}(L_1(P(v_{1,k}))) \cap \mathit{suffix}(L_2(P(v_{2,\ell})))$ such that $|y| > |z| - 1$. This contradicts that $z$ is a longest common suffix of $L_1(P(v_{1,i}))$ and $L_2(P(v_{2,j}))$, since $L_1(v_{1,i}) = L_2(v_{2,j})$. Hence $y \le |z| - 1$. If $v_{1,k}$ and $v_{2,\ell}$ are vertices satisfying $C_{k,\ell} = |z| - 1$, then $C_{i,j} = C_{k,\ell} + 1$. Note that such $v_{1,k}$ and $v_{2,\ell}$ always exist.
2. If $L_1(v_{1,i}) \ne L_2(v_{2,j})$, then trivially $\mathit{suffix}(L_1(P(v_{1,i}))) \cap \mathit{suffix}(L_2(P(v_{2,j}))) = \{\varepsilon\}$. Hence $C_{i,j} = 0$.

Consequently we obtain the following recurrence:

$$C_{i,j} = \begin{cases} 1 + \max(\{C_{k,\ell} \mid (v_{1,k}, v_{1,i}) \in E_1, (v_{2,\ell}, v_{2,j}) \in E_2\} \cup \{0\}) & \text{if } L_1(v_{1,i}) = L_2(v_{2,j}); \\ 0 & \text{otherwise.} \end{cases} \quad (3)$$

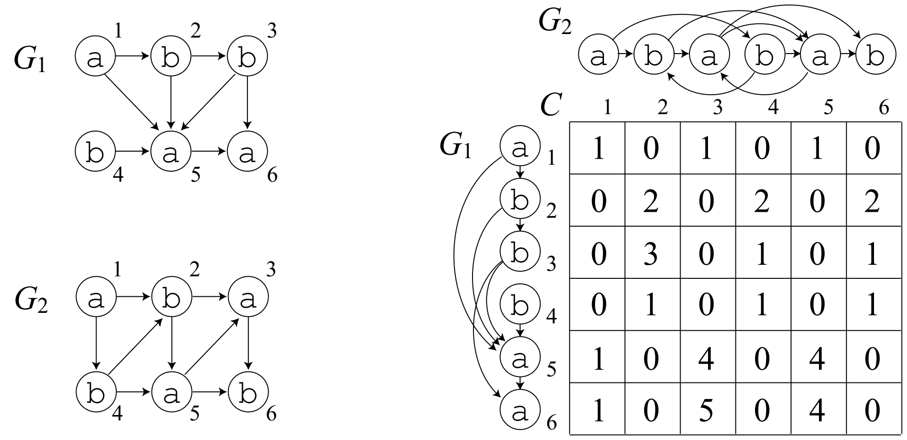



**Figure 2.** Example of dynamic programming for computing the length of a longest common substring of non-linear texts $G_1$ and $G_2$. Each vertex is annotated with its topological order. In this example, $\max C_{i,j} = 5$ and the longest common substring is `abbaa`.

We use dynamic programming to compute $C_{i,j}$ for all $1 \leq i \leq |V_1|$ and $1 \leq j \leq |V_2|$. Consider to compute $\max\{C_{k,\ell} \mid (v_{1,k}, v_{1,i}) \in E_1, (v_{2,\ell}, v_{2,j}) \in E_2\}$. For each fixed $(v_{1,k}, v_{1,i}) \in E_1$, we refer the value of $C_{k,\ell}$ for all $1 \leq \ell < j$ such that $(v_{2,\ell}, v_{2,j}) \in V_2$, in $O(|E_2|)$ time. Therefore, the total time complexity for computing $\max\{C_{k,\ell} \mid (v_{1,k}, v_{1,i}) \in E_1, (v_{2,\ell}, v_{2,j}) \in E_2\}$ is $O(|E_1||E_2|)$. Since we can sort vertices of $G_1$ and $G_2$ in topological ordering in linear time, the total time complexity is $O(|E_1||E_2|)$. The space complexity is clearly $O(|V_1||V_2|)$. □

An example of computing $C_{i,j}$ using dynamic programming is shown in Figure 2.

We remark that the recurrence of (3) is a natural generalization of that of (1) for computing the longest common substring of linear texts.

Furthermore, we can solve Problem 1 in case where *only one of the input non-linear texts is acyclic*:

**Theorem 3.** *If at least one of $G_1$ and $G_2$ is acyclic, then Problem 1 can be solved in $O(|E_1||E_2|)$ time and $O(|V_1||V_2|)$ space.*

*Proof.* Assume w.l.o.g. that $G_1$ is acyclic. Recall the proof of Theorem 2. A key observation is that it indeed suffices to sort one of the input non-linear texts in topological ordering.

For any vertex $v_{2,j} \in V_2$ and positive integer $h$, let $P_h(v_{2,j})$ denote the set of paths of length not greater than $h$, which end at vertex $v_{2,j}$. Assume we have sorted vertices of $G_1$. Let $C_{i,j}$ denote the length of a longest string in $\mathit{suffix}(L_1(P(v_{1,i}))) \cap \mathit{suffix}(L_2(P_r(v_{2,j})))$, where $r$ is the length of a longest path in $P(v_{1,i})$. We compute $C_{1,j}$ for each vertex $v_{2,j} \in V_2$ by: $C_{1,j} = 1$ if $L_1(v_{1,1}) = L_2(v_{2,j})$ and $C_{1,j} = 0$ otherwise. Then we compute $C_{i,j}$ for all $i > 1$ using the same recurrence as (3). Since the length of any element in $\mathit{substr}(G_1) \cap \mathit{substr}(G_2)$ is not greater than that of the longest path in $G_1$, $\max\{C_{i,j} \mid 1 \leq i \leq |V_1|, 1 \leq j \leq |V_2|\}$ equals to the length of a longest string in $\mathit{substr}(G_1) \cap \mathit{substr}(G_2)$. Consequently, $G_2$ does not have to be acyclic. □

A pseudo-code of our algorithm to solve the longest common substring problem for non-linear texts is shown in Algorithm 1.

**Algorithm 1**: Computing the length of longest common substring of non-linear texts.

**Input**: Acyclic non-linear text $G_1 = (V_1, E_1, L_1)$ and non-linear text $G_2 = (V_2, E_2, L_2)$.
**Output**: Length of a longest string in $substr(G_1) \cap substr(G_2)$.

```
topological sort G_1;
n ← |V_1|; m ← |V_2|;
Let C be an n × m integer array;
for i ← 1 to n do
    for j ← 1 to m do
        if f(v_{1,i}) = f(v_{2,j}) then
            C_{i,j} ← 1;
            forall v_{1,k} s.t. (v_{1,k}, v_{1,i}) ∈ E_1 do
                forall v_{2,ℓ} s.t. (v_{2,ℓ}, v_{2,j}) ∈ E_2 do
                    if C_{i,j} < 1 + C_{k,ℓ} then
                        C_{i,j} ← 1 + C_{k,ℓ};
        else
            C_{i,j} ← 0;
return max{C_{i,j} | 1 ≤ i ≤ n, 1 ≤ j ≤ m};
```

## 4 Computing Longest Common Subsequence Problem of Non-linear Texts

In this section, we tackle the problem of computing the length of longest common subsequence of two input non-linear texts. The problem is formalized as follows.

*Problem 4 (Longest common subsequence problem for non-linear texts).*

**Input:** Non-linear texts $G_1 = (V_1, E_1, L_1)$ and $G_2 = (V_2, E_2, L_2)$.
**Output:** The length of a longest string in $subseq(G_1) \cap subseq(G_2)$.

For example, see the non-linear texts $G_1$ and $G_2$ of Figure 3. The solution to the above problem is 4, since there is a longest common subsequence `acdb` of $G_1$ and $G_2$.

In the sequel we present our algorithm to solve the above problem in case where both $G_1$ and $G_2$ are acyclic.

**Theorem 5.** *If $G_1$ and $G_2$ are acyclic, then Problem 4 can be solved in $O(|E_1||E_2|)$ time and $O(|V_1||V_2|)$ space.*

*Proof.* Let $v_{1,i}$ and $v_{2,j}$ denote the $i$-th and $j$-th vertex in topological ordering in $G_1$ and in $G_2$, for $1 \le i \le |V_1|$ and $1 \le j \le |V_2|$, respectively. Let $C_{i,j}$ denote the length of a longest string in $subseq(L_1(P(v_{1,i}))) \cap subseq(L_2(P(v_{2,j})))$. $C_{i,j}$ can be calculated as follows.

1. If $L_1(v_{1,i}) = L_2(v_{2,j})$, there are two cases to consider:
   (a) If there are no arcs to $v_{1,i}$ or to $v_{2,j}$, i.e., $P(v_{1,i}) = \{v_{1,i}\}$ or $P(v_{2,j}) = \{v_{2,j}\}$, then clearly $C_{i,j} = 1$.
   (b) Otherwise, let $v_{1,k}$ and $v_{2,\ell}$ be any nodes s.t. $(v_{1,k}, v_{1,i}) \in E_1$ and $(v_{2,\ell}, v_{2,j}) \in E_2$, respectively. Let $z$ be a longest string in $subseq(L_1(P(v_{1,i}))) \cap subseq(L_2(P(v_{2,j})))$. Assume on the contrary that there exists a string $y \in subseq(L_1(P(v_{1,k}))) \cap subseq(L_2(P(v_{2,\ell})))$ such that $|y| > |z| - 1$. This contradicts that $z$ is a longest common subsequence of $L_1(P(v_{1,i}))$ and $L_2(P(v_{2,j}))$, since $L_1(v_{1,i}) = L_2(v_{2,j})$. Hence $|y| \le |z| - 1$. If $v_{1,k}$ and $v_{2,\ell}$ are vertices satisfying $C_{k,\ell} = |z| - 1$, then $C_{i,j} = C_{k,\ell} + 1$. Note that such $v_{1,k}$ and $v_{2,\ell}$ always exist.

2. If $L_1(v_{1,i}) \neq L_2(v_{2,j})$, there are two cases to consider:
   (a) If there are no arcs to $v_{1,i}$ and to $v_{2,j}$, i.e.,$P(v_{1,i}) = \{v_{1,i}\}$ and $P(v_{2,j}) = \{v_{2,j}\}$, then clearly $C_{i,j} = 0$.
   (b) Otherwise, let $v_{1,k}$ and $v_{2,\ell}$ be any nodes s.t. $(v_{1,k}, v_{1,i}) \in E_1$ and $(v_{2,\ell}, v_{2,j}) \in E_2$, respectively. Let $z$ be a longest string in $\mathit{subseq}(L_1(P(v_{1,i}))) \cap \mathit{subseq}(L_2(P(v_{2,j})))$. Assume on the contrary that there exists a string $y \in \mathit{subseq}(L_1(P(v_{1,k}))) \cap \mathit{subseq}(L_2(P(v_{2,j})))$ such that $|y| > |z|$. This contradicts that $z$ is a longest common subsequence of $L_1(P(v_{1,i}))$ and $L_2(P(v_{2,j}))$, since $\mathit{subseq}(L_1(P(v_{1,k}))) \cap \mathit{subseq}(L_2(P(v_{2,j}))) \subseteq \mathit{subseq}(L_1(P(v_{1,i}))) \cap \mathit{subseq}(L_2(P(v_{2,j})))$. Hence $|y| \leq |z|$. If $v_{1,k}$ is a vertex satisfying $C_{k,j} = |z|$, then $C_{i,j} = C_{k,j}$. Similarly, if $v_{2,\ell}$ is a vertex satisfying $C_{i,\ell} = |z|$, then $C_{i,j} = C_{i,\ell}$. Note that such $v_{1,k}$ or $v_{2,\ell}$ always exists.

Consequently we obtain the following recurrence:

$$C_{i,j} = \begin{cases} 1+\max(\{C_{k,\ell} \mid (v_{1,k}, v_{1,i}) \in E_1, (v_{2,\ell}, v_{2,j}) \in E_2\} \cup \{0\}) & \text{if } L_1(v_{1,i}) = L_2(v_{2,j}); \\ \max\begin{pmatrix} \{C_{k,j} \mid (v_{1,k}, v_{1,i}) \in E_1\} \cup \{C_{i,\ell} \mid (v_{2,\ell}, v_{2,j}) \in E_2\} \\ \cup\{0\} \end{pmatrix} & \text{otherwise.} \end{cases} \quad (4)$$

We use dynamic programming to compute $C_{i,j}$ for all $1 \leq i \leq |V_1|$ and $1 \leq j \leq |V_2|$.

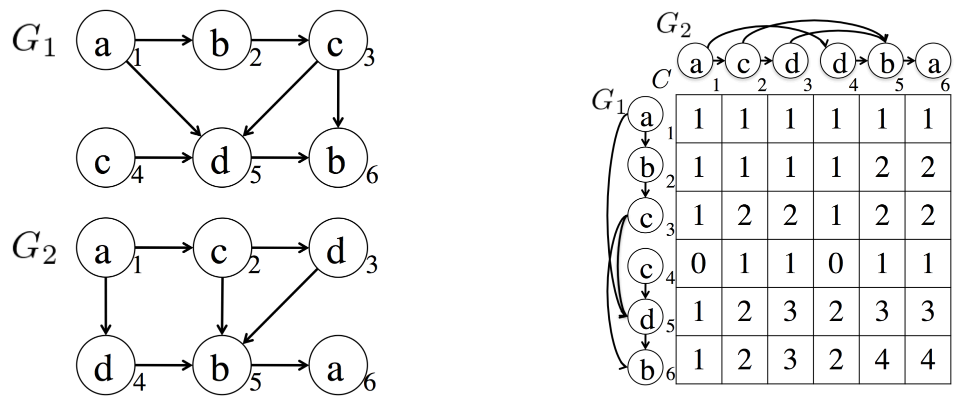
By similar arguments to the proof of Theorem 2, computing $\max\{C_{k,\ell} \mid (v_{1,k}, v_{1,i}) \in E_1, (v_{2,\ell}, v_{2,j}) \in E_2\}$ takes $O(|E_1||E_2|)$ time.

Consider to compute $\max\{C_{k,j}, C_{i,\ell} \mid (v_{1,k}, v_{1,i}) \in E_1, (v_{2,k}, v_{2,j}) \in E_2\}$. For each fixed $(v_{1,k}, v_{1,i}) \in E_1$, we refer the value of $C_{k,j}$ for all $1 \leq j \leq |V_2|$ in $O(|V_2|)$ time. Similarly, for each fixed $(v_{2,\ell}, v_{2,j}) \in E_2$, we refer the value of $C_{i,\ell}$ for all $1 \leq i \leq |V_1|$ in $O(|V_1|)$ time. Therefore, the total time cost for computing $\max\{C_{k,j}, C_{i,\ell} \mid (v_{1,k}, v_{1,i}) \in E_1, (v_{2,\ell}, v_{2,j}) \in E_2\}$ is $O(|V_2||E_1| + |V_1||E_2|)$.

Since we can sort vertices of $G_1$ and $G_2$ in topological ordering in linear time, the total time complexity is $O(|E_1||E_2|)$. The space complexity is clearly $O(|V_1||V_2|)$. □

An example of computing $C_{i,j}$ using dynamic programming is show in Figure 3. We remark that the recurrence of (4) is a natural generalization of that of (2) for computing the longest common subsequence of linear texts.

Algorithm 2 shows a pseudo-code of our algorithm to solve Problem 4 in case where both $G_1$ and $G_2$ are acyclic.




| $C$ | a (1) | c (2) | d (3) | d (4) | b (5) | a (6) |
|---|---|---|---|---|---|---|
| a (1) | 1 | 1 | 1 | 1 | 1 | 1 |
| b (2) | 1 | 1 | 1 | 1 | 2 | 2 |
| c (3) | 1 | 2 | 2 | 1 | 2 | 2 |
| c (4) | 0 | 1 | 1 | 0 | 1 | 1 |
| d (5) | 1 | 2 | 3 | 2 | 3 | 3 |
| b (6) | 1 | 2 | 3 | 2 | 4 | 4 |

**Figure 3.** Example of dynamic programming for computing the length of a longest common subsequence of non-linear texts $G_1$ and $G_2$. Each vertex is annotated with its topological order. In this example, $\max C_{i,j} = 4$ and the longest common subsequence is `acdb`.

**Algorithm 2**: Computing the length of longest common subsequence of acyclic non-linear texts

```
Input: Two acyclic non-linear texts G1 = (V1, E1, L1), G2 = (V2, E2, L2)
Output: Length of a longest string in subseq(G1) ∩ subseq(G2)
topological sort G1;
topological sort G2;
n ← |V1|; m ← |V2|;
Let C be an n × m integer array;
for i ← 1 to n do
    for j ← 1 to m do
        if f(v1,i) = f(v2,j) then
            C_{i,j} ← 1;
            forall v1,k s.t. (v1,k, v1,i) ∈ E1 do
                forall v2,ℓ s.t. (v2,ℓ, v2,j) ∈ E2 do
                    if C_{i,j} < 1 + C_{k,ℓ} then
                        C_{i,j} ← 1 + C_{k,ℓ};
        else
            C_{i,j} ← 0;
            forall v1,k s.t. (v1,k, v1,i) ∈ E1 do
                if C_{i,j} < C_{k,j} then
                    C_{i,j} ← C_{k,j};
            forall v2,ℓ s.t. (v2,ℓ, v2,j) ∈ E2 do
                if C_{i,j} < C_{i,ℓ} then
                    C_{i,j} ← C_{i,ℓ};
return max{C_{i,j} | 1 ≤ i ≤ n, 1 ≤ j ≤ m};
```

## 5 Computing Longest Common Subsequence of Cyclic Non-linear Texts

In this section, we present an algorithm to solve Problem 4 in case where the input non-linear texts are cyclic. We output $\infty$ if $subseq(G_1) \cap subseq(G_2)$ is infinite, and do the length of a longest string in $subseq(G_1) \cap subseq(G_2)$ otherwise.

We transform a cyclic non-linear text $G = (V, E, L)$ into an acyclic non-linear text $G' = (V', E', L')$ based on the strongly connected components. For each vertex $v \in V$, let $[v]$ denote the set of vertices that belong to the same strongly connected component. Formally, $G'$ is defined as

$$V' = \{[v] \mid v \in V\},$$
$$E' = \{([v], [u]) \mid [v] \neq [u], (v', u') \in E \text{ for some } v' \in [v],\ u' \in [u]\} \cup \{(v, v) \mid |[v]| \geq 2\},$$

and $L'([v]) = \{L(v) \mid v \in [v]\} \subseteq \Sigma$. We regard each $[v]$ as a single vertex that is contracted from vertices in $[v]$. Observe that $subseq(G') = subseq(G)$.

An example of transformed acyclic non-linear texts is shown in Figure 4.

**Theorem 6.** *If $G_1$ and/or $G_2$ are cyclic, then Problem 4 can be solved in $O(|E_1||E_2| + |V_1||V_2| \log |\Sigma|)$ time and $O(|V_1||V_2|)$ space.*

*Proof.* We first transform cyclic non-linear texts $G_1$ and $G_2$ into corresponding acyclic non-linear texts $G'_1$ and $G'_2$, as described previously. Let $v'_{1,i}$ and $v'_{2,j}$ denote the $i$-th and $j$-th vertex in topological ordering in $G'_1$ and $G'_2$, for $1 \leq i \leq |V'_1|$ and $1 \leq j \leq |V'_2|$, respectively. Let $S_1$ and $S_2$ denote the sets of vertices which has a loop, namely, $S_1 = \{L'_1(v'_{1,i}) \mid (v'_{1,i}, v'_{1,i}) \in E'_1\}$ and $S_2 = \{L'_2(v'_{2,j}) \mid (v'_{2,j}, v'_{2,j}) \in E'_2\}$. If $S_1 \cap S_2 \neq \emptyset$, then let $c$ be any character in $S_1 \cap S_2$. Clearly an infinite repetition $c^*$ of $c$ is a common subsequence of $G_1$ and $G_2$, and hence we output $\infty$.

In the sequel, consider the case where $S_1 \cap S_2 = \emptyset$. In this case, it is clear that $subseq(G_1) \cap subseq(G_2)$ is finite. Let $C_{i,j}$ denote the length of a longest string in $subseq(L'_1(P(v'_{1,i}))) \cap subseq(L'_2(P(v'_{2,j})))$. $C_{i,j}$ can be calculated as follows.

1. If $L'(v'_{1,i}) \cap L'(v'_{2,j}) \neq \emptyset$, there are two cases to consider:
   (a) If there are no arcs to $v'_{1,i}$ or to $v'_{2,j}$, i.e., $P(v'_{1,i}) = \{v'_{1,i}\}$ or $P(v'_{2,j}) = \{v'_{2,j}\}$, then clearly $C_{i,j} = 1$.
   (b) Otherwise, let $v'_{1,k}$ and $v'_{2,\ell}$ be any nodes s.t. $(v'_{1,k}, v'_{1,i}) \in E'_1$ and $(v'_{2,\ell}, v'_{2,j}) \in E'_2$, respectively. Let $z$ be a longest string in $subseq(L'_1(P(v'_{1,i}))) \cap subseq(L'_2(P(v'_{2,j})))$. Assume on the contrary that there exists a string $y \in subseq(L'_1(P(v'_{1,k}))) \cap subseq(L'_2(P(v'_{2,\ell})))$ such that $|y| > |z| - 1$. This contradicts that $z$ is a longest common subsequence of $L'_1(P(v'_{1,i}))$ and $L'_2(P(v'_{2,j}))$, since $L'_1(v'_{1,i}) \cap L'_2(v'_{2,j}) \neq \emptyset$. Hence $|y| \leq |z| - 1$. If $v'_{1,k}$ and $v'_{2,\ell}$ are vertices satisfying $C_{k,\ell} = |z| - 1$, then $C_{i,j} = C_{k,\ell} + 1$. Note that such $v'_{1,k}$ and $v'_{2,\ell}$ always exist.
2. If $L'(v'_{1,i}) \cap L'(v'_{2,j}) = \emptyset$, there are two cases to consider:
   (a) If there are no arcs to $v'_{1,i}$ and to $v'_{2,j}$, i.e., $P(v'_{1,i}) = \{v'_{1,i}\}$ and $P(v'_{2,j}) = \{v'_{2,j}\}$, then clearly $C_{i,j} = 0$.
   (b) Otherwise, let $v'_{1,k}$ and $v'_{2,\ell}$ be any nodes s.t. $(v'_{1,k}, v'_{1,i}) \in E'_1$ and $(v'_{2,\ell}, v'_{2,j}) \in E'_2$, respectively. Let $z$ be a longest string in $subseq(L'_1(P(v'_{1,i}))) \cap subseq(L'_2(P(v'_{2,j})))$. Assume on the contrary that there exists a string $y \in subseq(L'_1(P(v'_{1,k}))) \cap subseq(L'_2(P(v'_{2,j})))$ such that $|y| > |z|$. This contradicts that $z$ is a longest common subsequence of $L'_1(P(v'_{1,i}))$ and $L'_2(P(v'_{2,j}))$, since $subseq(L'_1(P(v'_{1,k}))) \cap$

$subseq(L'_2(P(v'_{2,j}))) \subseteq subseq(L'_1(P(v'_{1,i}))) \cap subseq(L'_2(P(v'_{2,j})))$. Hence $|y| \leq |z|$. If $v'_{1,k}$ is a vertex satisfying $C_{k,j} = |z|$, then $C_{i,j} = C_{k,j}$. Similarly, if $v'_{2,\ell}$ is a vertex satisfying $C_{i,\ell} = |z|$, then $C_{i,j} = C_{i,\ell}$. Note that such $v'_{1,k}$ ($k \neq i$) or $v'_{2,\ell}$ ($\ell \neq j$) always exists.

Consequently we obtain the following recurrence:

$$C_{i,j} = \begin{cases} 1+\max(\{C_{k,\ell} \mid (v'_{1,k}, v'_{1,i}) \in E'_1, (v'_{2,\ell}, v'_{2,j}) \in E'_2\} \cup \{0\}) & \text{If } L'(v'_{1,i}) \cap L'(v'_{2,j}) \neq \emptyset; \\ \max\begin{pmatrix} \{C_{k,j} \mid (v'_{1,k}, v'_{1,i}) \in E_1\} \cup \{C_{i,\ell} \mid (v'_{2,\ell}, v'_{2,j}) \in E_2\} \\ \cup\{0\} \end{pmatrix} & \text{otherwise.} \end{cases} \quad (5)$$

It is well-known that we can transform $G_1$ and $G_2$ into $G'_1$ and $G'_2$ in linear time, based on strongly connected components.

For each self-loop such as $(v'_{1,i}, v'_{1,i}) \in E_1$, we refer the value of $C_{i,j}$ for all $1 \leq j \leq |V'_2|$ in $O(|V'_2|)$ time. Similarly, for each self-loop such as $(v'_{2,j}, v'_{2,j}) \in E_2$, we refer the value of $C_{i,j}$ for all $1 \leq i \leq |V'_1|$ in $O(|V'_1|)$ time. For the other arcs, we can compute $C_{i,j}$ for all $1 \leq i \leq |V'_1|$ and $1 \leq j \leq |V'_2|$ using dynamic programming in $O(|E'_1| \cdot |E'_2|)$ time, in a similar way as the previous section. Therefore the total time cost for computing $C_{i,j}$ is $O(|E'_1| \cdot |E'_2|)$.

Let $\Sigma_1$ and $\Sigma_2$ be the sets of characters that appear in $G_1$ and $G_2$, respectively. The time cost to compute $S_1 \cap S_2$ is $O(|\Sigma_1| \log |\Sigma_2| + |\Sigma_2| \log |\Sigma_1|)$ using a balanced tree. Assume $S_1 \cap S_2 = \emptyset$, and consider to compute $L'(v'_{1,i}) \cap L'(v'_{2,j})$. If $|L'(v'_{1,i})| > 1$ and $|L'(v'_{2,j})| > 1$, then we know $L'(v'_{1,i}) \cap L'(v'_{2,j}) = \emptyset$ since $S_1 \cap S_2 = \emptyset$. If $|L'(v'_{1,i})| = 1$ and/or $|L'(v'_{2,j})| = 1$, then $L'(v'_{1,i}) \cap L'(v'_{2,j})$ can be computed in $O(\log |\Sigma|)$ time using a balanced tree, where $|\Sigma| = \max\{|\Sigma_1|, |\Sigma_2|\}$. Therefore the total time cost to compare $L'(v'_{1,i})$ and $L'(v'_{2,j})$ for all $1 \leq i \leq |V'_1|$ and $1 \leq j \leq |V'_2|$ is $O(|V'_1||V'_2| \log |\Sigma|)$. The total time complexity becomes $O(|E_1|+|E_2|+|E'_1||E'_2|+|V'_1||V'_2| \log |\Sigma|+|\Sigma_1| \log |\Sigma_2|+|\Sigma_2| \log |\Sigma_1|) = O(|E_1||E_2| + |V_1||V_2| \log |\Sigma|)$, since $|\Sigma_1| \leq |V_1|$ and $|\Sigma_2| \leq |V_2|$. The total space complexity is $O(|V'_1||V'_2| + |\Sigma_1| \log |\Sigma_2| + |\Sigma_2| \log |\Sigma_1|) = O(|V_1||V_2|)$. □

An example of computing $C_{i,j}$ using dynamic programming is shown in Figure 4. A pseudo-code of our algorithm is shown in Algorithm 3.

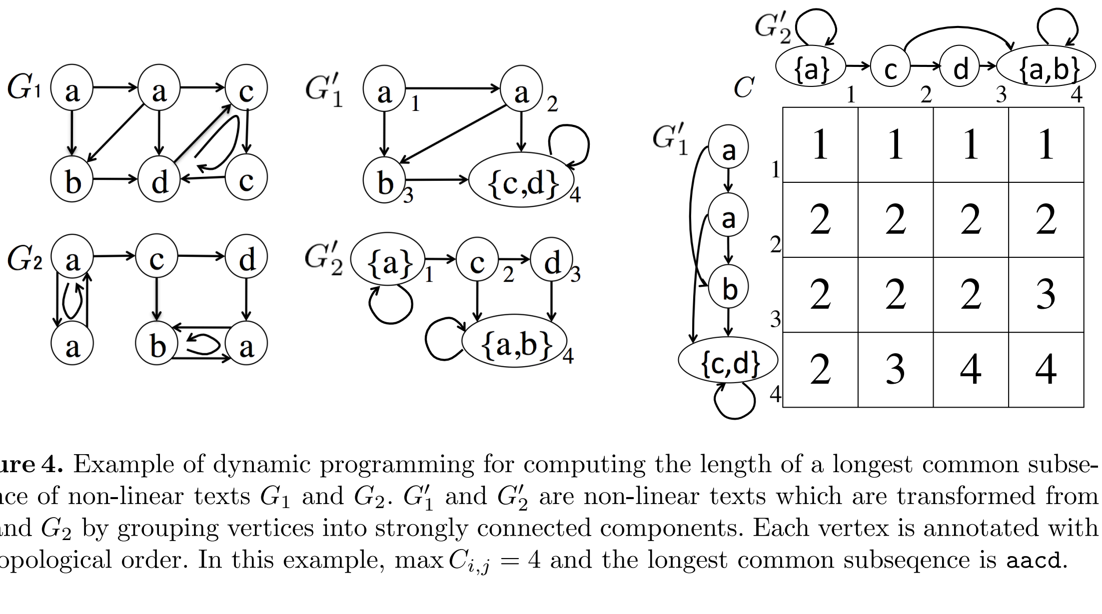



**Figure 4.** Example of dynamic programming for computing the length of a longest common subsequence of non-linear texts $G_1$ and $G_2$. $G'_1$ and $G'_2$ are non-linear texts which are transformed from $G_1$ and $G_2$ by grouping vertices into strongly connected components. Each vertex is annotated with its topological order. In this example, $\max C_{i,j} = 4$ and the longest common subseqence is `aacd`.

**Algorithm 3**: Computing the length of longest common subsequence of cyclic non-linear texts

```
Input: Two non-linear texts G_1 = (V_1, E_1, L_1), G_2 = (V_2, E_2, L_2)
Output: Length of a longest string in subseq(G_1) ∩ subseq(G_2)
G'_1 ← Strongly Connected Components G_1;
G'_2 ← Strongly Connected Components G_2;
Let S_1 be a set of vertices which belong to cycles in G_1;
Let S_2 be a set of vertices which belong to cycles in G_2;
if S_1 ∩ S_2 ≠ ∅ then
    return ∞ ;
else
    topological sort G'_1;
    topological sort G'_2;
    Let C be an |V'_1| × |V'_2| integer array;
    for i ← 1 to |V'_1| do
        for j ← 1 to |V'_2| do
            if (v'_{1,i}, v'_{1,i}) ∈ E'_1 then
                if (v'_{2,j}, v'_{2,j}) ∈ E'_2 then
                    C_{i,j} ←Vertex-mismatch (v'_{1,i}, v'_{2,j});
                else if L(v'_{1,i}) ⊇ L(v'_{2,j}) then
                    C_{i,j} ←Vertex-match (v'_{1,i}, v'_{2,j});
                else
                    C_{i,j} ←Vertex-mismatch(v'_{1,i}, v'_{2,j});
            else if (v'_{2,j}, v'_{2,j}) ∈ E'_2 then
                if L(v'_{1,i}) ⊆ L(v'_{2,j}) then
                    C_{i,j} ←Vertex-match(v'_{1,i}, v'_{2,j});
                else
                    C_{i,j} ←Vertex-mismatch(v'_{1,i}, v'_{2,j});
            else if L(v'_{1,i}) = L(v'_{2,j}) then
                C_{i,j} ←Vertex-match(v'_{1,i}, v'_{2,j});
            else
                C_{i,j} ←Vertex-mismatch(v'_{1,i}, v'_{2,j});
return max{C_{i,j} | 1 ≤ i ≤ |V'_1|, 1 ≤ j ≤ |V'_2|};
```

**Algorithm 4**: Vertex-match$(v_{1,i}, v_{2,j})$

```
C_{i,j} ← 1
forall v_{1,k} s.t. (v_{1,k}, v_{1,i}) ∈ E_1 do
    forall v_{2,ℓ} s.t. (v_{2,ℓ}, v_{2,j}) ∈ E_2 do
        if C_{i,j} < 1 + C_{k,ℓ} then
            C_{i,j} ← 1 + C_{k,ℓ}
return C_{i,j}
```

**Algorithm 5**: Vertex-mismatch($v_{1,i}, v_{2,j}$)

$C_{i,j} \leftarrow 0$
**forall** $v_{1,k}$ *s.t.* $(v_{1,k}, v_{1,i}) \in E_1$ **do**
    **if** $C_{i,j} < C_{k,j}$ **then**
        $C_{i,j} \leftarrow C_{k,j}$
**forall** $v_{2,\ell}$ *s.t.* $(v_{2,\ell}, v_{2,j}) \in E_2$ **do**
    **if** $C_{i,j} < C_{i,\ell}$ **then**
        $C_{i,j} \leftarrow C_{i,\ell}$
**return** $C_{i,j}$

## 6 Conclusions

We considered the longest common substring and subsequence problems between two non-linear texts. We showed that when the texts are acyclic, the problem can be solved in $O(|E_1||E_2|)$ time and $O(|V_1||V_2|)$ space by a dynamic programming approach. Furthermore, we extend our algorithm and consider the case where the texts can contain cycles, and presented an $O(|E_1||E_2| + |V_1||V_2| \log |\Sigma|)$ time and $O(|V_1||V_2|)$ space algorithm for the longest common subsequence problem. The longest common substring between general graphs is an open problem.

## References


1. T. Akutsu: *A linear time pattern matching algorithm between a string and a tree*, in Proc. CPM'93, 1993, pp. 1–10.
2. A. Amir, M. Lewenstein, and N. Lewenstein: *Pattern matching in hypertext*, in Proc. WADS'97, vol. 1272 of LNCS, 1997, pp. 160–173.
3. U. Manber and S. Wu: *Approximate string matching with arbitrary costs for text and hypertext*, in Proc. IAPR, 1992, pp. 22–33.
4. M. Mohri: *Edit-distance of weighted automata: General definitions and algorithms.* Int. J. Found. Comput. Sci, 14(6) 2003, pp. 957–982.
5. G. Navarro: *Improved approximate pattern matching on hypertext.* Theoretial Computer Science, 237(1–2) 2000, pp. 455–463.
6. K. Park and D. K. Kim: *String matching in hypertext*, in Proc. CPM'95, 1995, pp. 318–329.
7. D. Q. Thang: *Algorithm to determine longest common subsequences of two finite languages*, in New Challenges for Intelligent Information and Database System, vol. 351/2011 of Studies in Computational Intelligence, 2011, pp. 3–12.