Journal of Computer Science and Cybernetics, V.35, N.1 (2019), 21–37
DOI 10.15625/1813-9663/35/1/13293

AUTOMATA TECHNIQUE FOR THE LCS PROBLEM

NGUYEN HUY TRUONG

School of Applied Mathematics and Informatics, Hanoi University of Science and Technology, Vietnam; truong.nguyenhuy@hust.edu.vn

Abstract. In this paper, we introduce two efficient algorithms in practice for computing the length of a longest common subsequence of two strings, using automata technique, in sequential and parallel ways. For two input strings of lengths m and n with $m \leq n$, the parallel algorithm uses k processors ($k \leq m$) and costs time complexity $O(n)$ in the worst case, where k is an upper estimate of the length of a longest common subsequence of the two strings. These results are based on the Knapsack Shaking approach proposed by P. T. Huy et al. in 2002. Experimental results show that for the alphabet of size 256, our sequential and parallel algorithms are about 65.85 and $3.41m$ times faster than the classical dynamic programming algorithm proposed by Wagner and Fisher in 1974, respectively.

Keywords. Automata; Dynamic programing; Knapsack shaking approach; Longest common subsequence; Parallel LCS.

1 INTRODUCTION

The longest common subsequence (LCS) problem is a well-known problem in computer science [2, 3, 7, 8] and has many applications [1, 8, 14], especially in approximate pattern matching [8, 10, 12]. In 1972, authors V. Chvatal, D. A. Klarner and D. Knuth listed the problem of finding the longest common subsequence of the two strings in 37 selected combinatorial research problems [3]. The LCS problem for k strings ($k > 2$) is the NP-hard problem [7, 9, 11].

For the approximate pattern matching problem, the length of a longest common subsequence of two strings is used to compute the similarity between the two strings [10, 12]. Our work is concerned with the problem of finding the length of a longest subsequence of two strings of lengths m and n. In addition, our main objective is planning to deal with the approximate search problem in the future. So, we will assume that $m \leq n$, where the pattern of length m and the text of length n.

In 1974, Wagner and Fischer proposed one of the first algorithms to solve the LCS problem for two strings. This algorithm is based on dynamic programming approach with the worst case time complexity $O(mn)$ and considered as a classical algorithm for the LCS problem (hereafter called the Algorithm WF) [2, 4, 5, 8, 13, 14, 16]. A list of existing sequential algorithms for the LCS problem and a theoretical comparison of them could be found in [8]. Furthermore, to compute the length of a longest common subsequence of two strings effectively, many parallel algorithms have been made [4, 13, 15, 16]. According to Xu et al. [15], their parallel algorithm, which uses k processors for $1 \leq k \leq \max\{m, n\}$ and costs time complexity $O(mn/k)$ in the worst case, is the fastest and cost optimal parallel algorithm for

LCS problem. Almost these algorithms including sequential as well as parallel algorithms have been developed from the Algorithm WF [4, 8, 13, 15, 16].

The goal of this paper is to develop algorithms in practice. In [8], the authors have suggested that the finite automata approach will be the best choice to solve the LCS problem. In this paper, based on the Knapsack Shaking approach introduced by P. T. Huy et al. in 2002 that is also a finite automata technique [6], we propose two efficient algorithms in practice for computing the length of a longest common subsequence of two strings in sequential and parallel ways. The time complexity of the parallel algorithm uses k processors ($k \leq m$) and costs time complexity $O(n)$ in the worst case, where k is an upper estimate of the length of a longest common subsequence of the two strings. Because of our assumption that $m \leq n$, on the theoretical side, our parallel algorithm is better than the Xu et al.'s parallel algorithm.

In our experiments, we only compute the length of a longest common subsequence of two strings and compare our two algorithms with the Algorithm WF. Note that the Algorithm WF is not fast, but it is simple and classical in the field of the longest common subsequence. Hence, we consider the running time of the Algorithm WF is as a standard unit of measurement for the running time of our algorithms. Experimental results show that for the alphabet of size 256, our sequential and parallel algorithms are about 65.85 and $3.41m$ times faster than the Algorithm WF, respectively.

The rest of the paper is organized as follows. In Section 2, we recall some basic notations, concepts and facts in [6, 14, 16] which will be used in the sequel. Section 3 constructs mathematical basis for the development of automata technique to design two sequential and parallel algorithms for the LCS problem. The experimental results comparing our algorithms with the Algorithm WF are shown in the tables in Section 4. Finally, in Section 5, we draw some conclusions from our automata technique and experimental results.

2 PRELIMINARIES

Let Σ be a finite set which we call an alphabet. The size of Σ is the number of elements belonging to Σ, denoted by $|\Sigma|$. An element of Σ is called a letter. A string p of length m on the alphabet Σ is a finite sequence of letters of Σ and we write

$$p = p[1]p[2]\dots p[m],\ p[i] \in \Sigma,\ 1 \leq i \leq m,$$

where m is a positive integer. The length of the string p is the number of letters in it, denoted by $|p|$. A special string having no letters is called empty string, denoted by ϵ.

Notice that for the string $p = p[1]p[2]\dots p[m]$, we can write $p = p[1..m]$ in short.

The notation Σ^* denotes the set of all strings on the alphabet Σ. The operator of strings is concatenation that joins strings end to end. The concatenation of the two strings u and v is denoted by uv.

Let s be a string. If $s = uv$ for some strings u and v, then the string u is called a prefix of the string s.

Now, we will restate the LCS problem.

Definition 1 ([16]). Let p be a string of length m and u be a string over the alphabet Σ. Then u is a subsequence of p if there exists a integer sequence $j_1, j_2, \dots, j_t$ such that $1 \leq j_1 < j_2 < \dots < j_t \leq m$ and $u = p[j_1]p[j_2]\dots p[j_t]$.

Definition 2 ([16]). Let p be a string of length m and u be a string over the alphabet Σ. Then u is a common subsequence of p and s if u is a subsequence of p and a subsequence of s.

Definition 3 ([16]). Let p, s and u be strings over the alphabet Σ. Then u is a longest common subsequence of p and s if two following conditions are satisfied.

(i) u is a subsequence of p and s.

(ii) There does not exist a common subsequence v of p and s such that $|v| > |u|$.

We use the notation $LCS(p, s)$ to denote an arbitrary longest common subsequence of p and s. The length of a $LCS(p, s)$ is denoted by $lcs(p, s)$.

By convention if two strings p and s does not have any longest common subsequences, then the $lcs(p, s)$ is considered to equal 0.

Example 4. Let $p = bgcadb$ and $s = abhcbad$. Then string $bcad$ is a $LCS(p, s)$ and $lcs(p, s) = 4$.

Let p and s be two strings of lengths m and n over the alphabet $\Sigma, m \leq n$. The LCS problem is given in two following forms [6]:

Problem 1: Find a $LCS(p, s)$.

Problem 2: Compute the $lcs(p, s)$.

To illustrate the simple way to solve the LCS problem, we use the Algorithm WF. To find a $LCS(p, s)$ and compute the $lcs(p, s)$, the Algorithm WF defines a dynamic programming matrix $L(m, n)$ recursively as follows [14].

$$L(i,j) = \begin{cases} 0 & i = 0 \text{ or } j = 0, \\ L(i-1, j-1) + 1 & p[i] = s[j], \\ \max\{L(i, j-1), L(i-1, j)\} & \text{otherwise}, \end{cases}$$

where $L(i, j)$ is the $lcs(p[1..i], s[1..j])$ for $1 \leq i \leq m$, $1 \leq j \leq n$.

Example 5. Let $p = bgcadb$ and $s = abhcbad$. Use the Algorithm WF, we obtain the $L(m, n)$ below. Then $lcs(p, s) = L(6, 7) = 4$. In Table 1, by traceback procedure, starting from value 4 back to value 1, we get a $LCS(p, s)$ to be a string $bcad$.

Table 1. The dynamic programming matrix L

	$s =$		a	b	h	c	b	a	d
p =	i, j	0	1	2	3	4	5	6	7
	0	0	0	0	0	0	0	0	0
b	1	0	0	1	1	1	1	1	1
g	2	0	0	1	1	1	1	1	1
c	3	0	0	1	1	2	2	2	2
a	4	0	1	1	1	2	2	3	3
d	5	0	1	1	1	2	2	3	4
b	6	0	1	2	2	2	3	3	4

Next, we recall important concepts in [6].

Definition 6 ([6]). Let $u = p[j_1]p[j_2]\dots p[j_t]$ be a subsequence of p. Then an element of the form $(j_1, j_2, \dots, j_t)$ is called a location of u in p.

From Definition 6 we know that the subsequence u may have many different locations in p. If all the different locations of u are arranged in the dictionary order, then we call the least element to be the leftmost location of u, denoted by $LeftID(u)$. We denote by $Rm(u)$ the last component in $LeftID(u)$ [6].

Example 7. Let $p = aabcadabcd$ and $u = abd$. Then u is a subsequence of p and has seven different locations in p, in the dictionary order they are

$$(1,3,6), (1,3,10), (1,8,10), (2,3,6), (2,3,10), (5,8,10), (7,8,10).$$

It follows that $LeftID(u) = (1,3,6)$ and $Rm(u) = 6$.

Definition 8 ([6]). Let p be a string of length m. Then a configuration C of p is defined as follows.

1. Or C is the empty set. Then C is called the empty configuration of p and denoted by C_0.

2. Or $C = \{x_1, x_2, \dots, x_t\}$ is an ordered set of t subsequences of p for $1 \leq t \leq m$ such that the two following conditions are satisfied.

(i) $\forall i, 1 \leq i \leq t, |x_i| = i$,

(ii) $\forall x_i, x_j \in C$, if $|x_i| > |x_j|$, then $Rm(x_i) > Rm(x_j)$.

Set of all the configurations of p is denoted by Config(p).

Definition 9 ([6]). Let p be a string of length m on the alphabet Σ, $C \in$ Config(p) and $a \in \Sigma$. Then a state transition function φ on Config(p) $\times \Sigma$, φ : Config(p) $\times \Sigma \to$ Config(p), is defined as follows.

1. $\varphi(C, a) = C$ if $a \notin p$.

2. $\varphi(C_0, a) = \{a\}$ if $a \in p$.

3. Set $C' = \varphi(C, a)$. Suppose $a \in p$ and $C = \{x_1, x_2, \dots, x_t\}$ for $1 \leq t \leq m$. Then C' is determined by a loop using the loop control variable i whose value is changed from t down to 0:

a) For $i = t$, if the letter a appears at a location *index* in p such that *index* is greater than $Rm(x_t)$, then $x_{t+1} = x_t a$;

b) Loop from $i = t - 1$ down to 1, if the letter a appears at a location *index* in p such that $index \in (Rm(x_i), Rm(x_{i+1}))$, then $x_{i+1} = x_i a$;

c) For $i = 0$, if the letter a appears at a location *index* in p such that *index* is smaller than $Rm(x_1)$, then $x_1 = a$;

d) $C' = C$.

4. To accept an input string, the state transition function φ is extended as follows

$$\varphi : \text{Config}(p) \times \Sigma^* \to \text{Config}(p)$$

such that $\forall C \in$ Config(p), $\forall u \in \Sigma^*$, $\forall a \in \Sigma$, $\varphi(C, au) = \varphi(\varphi(C, a), u)$ and $\varphi(C, \epsilon) = C$.

Example 10. Let $p = bacdabcad$ and $C = \{c, ad, bab\}$. Then C is a configuration of p and $C' = \varphi(C, a) = \{a, ad, ada, baba\}$.

In 2002, P. T. Huy et al. introduced a method to solve the Problem 1 by using the automaton given as in the following theorem. In this way, they named their method the Knapsack Shaking approach [6].

Theorem 11 ([6]). *Let p and s be two strings of lengths m and n over the alphabet $\Sigma, m \leq n$. Let $\mathcal{A}_p = (\Sigma, Q, q_0, \varphi, F)$ corresponding to p be an automaton over the alphabet Σ, where*

- *The set of states $Q = Config(p)$,*
- *The initial state $q_0 = C_0$,*
- *The transition function φ is given as in Definition 9,*
- *The set of final states $F = \{C_n\}$, where $C_n = \varphi(q_0, s)$,*

Suppose $C_n = \{x_1, x_2, \ldots, x_t\}$ for $1 \leq t \leq m$. Then

1. For every subsequence u of p and s, there exists $x_i \in C_n, 1 \leq i \leq t$ such that the two following conditions are satisfied.

(i) $|u| = |x_i|$,

(ii) $Rm(x_i) \leq Rm(u)$.

2. A $LCS(p, s)$ equals x_t.

3 MAIN RESULTS

In this section, we propose a variant of Theorem 11 in general case (Theorem 12), construct mathematical basis based on Theorem 12 for the development of automata technique for the Problem 2 (Definition 22 and Theorem 25). Finally, we introduce two automata models (Theorems 35 and 39) to design two corresponding algorithms (Algorithms 1 and 2) for the Problem 2, discuss the time complexity of parallel algorithm (Proposition 40) and give some effective features of our algorithms in practice (Remarks 36 and 41).

In fact, when apply the Problem 2 to the approximate pattern matching problem, we only need to find a common subsequence of two strings such that the length of this common subsequence is equal to a given constant [10]. So, in general case, we replace the Theorem 11 with the following theorem. It is a variant of Theorem 11.

Theorem 12. *Let p and s be two strings of lengths m and n over the alphabet $\Sigma, m \leq n$. Let c be a positive integer constant, $1 \leq c \leq m$ and $\mathcal{A}_p^c = (\Sigma, Q, q_0, \varphi, F)$ corresponding to p be an automaton over the alphabet Σ, where*

- *The set of states $Q = Config(p)$,*
- *The initial state $q_0 = C_0$,*
- *The transition function φ is given as in Definition 9,*
- *The set of final states $F = \{C_f || C_f \in Config(p), C_f = \{x_1, x_2, \ldots, x_c\}$ or $C_f = \varphi(C_0, s)\}$.*

Suppose $C_f = \{x_1, x_2, \ldots, x_t\}$ is a final state for $1 \leq t \leq m$. Then there exists a substring u of s such that a $LCS(p, u)$ equals x_t.

Proof. If C_f is of the form $\varphi(C_0, s)$, then a $LCS(p, s)$ equals $x_t, 1 \leq t \leq m$ by Theorem 11, hence $u = s$. Conversely, the configuration C_f of the form $\{x_1, x_2, \ldots, x_t\}$ for $t = c$ then $\exists u$ is a prefix of s such that $C_f = \varphi(C_0, u)$ by Definition 9. By an application of Theorem 11 with two strings p and u, a $LCS(p, u)$ equals x_t. So, we complete the proof. ∎

Now, based on Theorem 12, we construct the mathematical basis for the development of automata technique for the Problem 2.

Definition 13. Let u be a subsequence of p. Then the weight of u in p, denoted by $W(u)$, is determined by the formula $W(u) = |p| + 1 - Rm(u)$.

Example 14. Let $p = aabcadabcd$ and $u = abd$. Then u is a subsequence of p and $W(u) = 5$.

Definition 15. Let p be a string of length m and C be a configuration of p. Then the weight of C is a ordered set, denoted by $W(C)$, and is determined as follows.

1. If $C = C_0$, then $W(C)$ is the empty set, denoted by W_0.
2. If $C = \{x_1, x_2, \ldots, x_t\}$ for $1 \leq t \leq m$, then $W(C) = \{W(x_1), W(x_2), \ldots, W(x_t)\}$.

Set of all the weights of all the configurations of p is denoted by WConfig(p).

Example 16. Let $p = abcadbad$ and $C = \{a, ba, bad\}$. Then C is a configuration of p and $W(C) = \{8, 5, 4\}$.

Definition 17. Let p be a string of length m, a be a letter of p and i be a location of a in p, $1 \leq i \leq m$. Then the weight of a at the location i in p, denoted by $W^i(a)$, is determined by the formula $W^i(a) = m + 1 - i$.

By convention if a is a letter of p and $a \neq p[i], 1 \leq i \leq m$, then the $W^i(a)$ is considered to equal 0.

Remark 18. Each letter of p at different locations has different weights. Assume that the letter a appears at two locations in p which are i and j, $i < j$. Then $W^i(a) > W^j(a)$ and say that the letter a at location i is heavier than at location j. If i is the lowest location, it means that i is the smallest index of p, such that $a = p[i]$, then the heaviest weight of a in p is equal to $W^i(a)$, denoted by $Wm(a)$.

Example 19. Let $p = aabcadabcd$. Then $W^1(a) = 10$, $W^7(a) = 4$. We say that the weight of a at location 1 in p is greater than at location 7 in p.

Set of all the letters in p is called the alphabet of p, denoted by Σ_p.

Definition 20. Let p be a string of length m. Then Ref of p is a function Ref : $\{1, \ldots, m\} \times \Sigma_p \to \{1, \ldots, m-1\}$ defined by the following formula

$$\text{Ref}(i, a) = \begin{cases} 0 & i = 1, \\ \max\{W^j(a) | W^j(a) < i \text{ for } m + 1 - i < j \leq m\} & 2 \leq i \leq m, \end{cases}$$

where $a \in \Sigma_p$.

Example 21. Let $p = bacdabcad$. Then the Ref of p is determined as in Table 2.

Table 2. The Ref of $p = bacdabcad$

Ref	a	b	c	d
1	0	0	0	0
2	0	0	0	1
3	2	0	0	1
4	2	0	3	1
5	2	4	3	1
6	5	4	3	1
7	5	4	3	6
8	5	4	7	6
9	8	4	7	6

Definition 22. Let p be a string of length m on the alphabet Σ, $W \in \mathrm{WConfig}(p)$ and $a \in \Sigma$. Then a state transition function δ on $\mathrm{WConfig}(p) \times \Sigma$, $\delta : \mathrm{WConfig}(p) \times \Sigma \to \mathrm{WConfig}(p)$, is defined as follows.

1. $\delta(W, a) = W$ if $a \notin p$.
2. $\delta(W_0, a) = \{\, Wm(a)\}$ if $a \in p$.
3. Set $W' = \delta(W, a)$. Suppose $a \in p$ and $W = \{w_1, w_2, \ldots, w_t\}$ for $1 \le t \le m$. Then W' is determined by a loop using the loop control variable i whose value is changed from t down to 0:

a) For $i = t$, if $\mathrm{Ref}(w_t, a) \neq 0$, then $w_{t+1} = \mathrm{Ref}(w_t, a)$;
b) Loop from $i = t - 1$ down to 1, if $\mathrm{Ref}(w_i, a) > w_{i+1}$, then $w_{i+1} = \mathrm{Ref}(w_i, a)$;
c) For $i = 0$, if $Wm(a) > w_1$, then $w_1 = Wm(a)$;
d) $W' = W$.

4. To accept an input string, the state transition function δ is extended as follows

$$\delta : \mathrm{WConfig}(p) \times \Sigma^* \to \mathrm{WConfig}(p)$$

such that $\forall W \in \mathrm{WConfig}(p),\ \forall u \in \Sigma^*,\ \forall a \in \Sigma,\ \delta(W, au) = \delta(\delta(W, a), u)$ and $\delta(W, \epsilon) = W$.

Example 23. Let $p = bacdabcad$ and $C = \{c, ad, bab\}$. Then C is a configuration of p. Set $W = W(C)$, then $W = \{7, 6, 4\}$ and $W' = \delta(W, a) = \{8, 6, 5, 2\}$.

Lemma 24. *Let p be a string of length m on the alphabet Σ, $C \in Config(p)$ and $a \in \Sigma$. Then $\delta(W(C), a) = W(\varphi(C, a))$, where δ and φ are given as in Definitions 22 and 9, respectively.*

Proof. Case $a \notin p$, then $\delta(W(C), a) = W(C) = W(\varphi(C, a))$ by Definitions 22 and 9.
Case $a \in p$, then $\delta(W(C_0), a) = \{\, Wm(a)\} = W(\{a\}) = W(\varphi(C, a))$ by Definitions 15, 22, 9 and Remark 18.
Case $a \in p$ and $C = \{x_1, x_2, \ldots, x_t\}$ for $1 \le t \le m$. Then $W(C) = \{W(x_1), W(x_2), \ldots, W(x_t)\}$. By Definitions 22 and 9, $\delta(W(C), a)$ and $\varphi(C, a)$ are both determined by a loop using the loop control variable i whose value is changed from t down to 0:

a) For $i = t$, if the letter a appears at a location *index* in p such that *index* is greater than $Rm(x_t)$, this is equivalent to $\mathrm{Ref}(W(x_t), a) \neq 0$ by Defintion 20, then $\varphi(C, a) =$

$\{x_1, x_2, \ldots, x_t, x_t a\}$ and $\delta(W(C), a) = \{W(x_1), W(x_2), \ldots, W(x_t), \text{Ref}(W(x_t), a)\}$. By Definitions 13 and 20, $W(x_t a) = \text{Ref}(W(x_t), a)\}$;

b) Loop from $i = t - 1$ down to 1, if the letter a appears at a location *index* in p such that *index* $\in (Rm(x_i), Rm(x_{i+1}))$, this is equivalent to $\text{Ref}(W(x_i), a) > W(x_{i+1})$ by Defintion 20, then

$$\varphi(C, a) = \{x_1, x_2, \ldots, x_i, x_i a, x_{i+2}, \ldots, x_t\} \text{ and}$$

$$\delta(W(C), a) = \{W(x_1), W(x_2), \ldots, W(x_i), \text{Ref}(W(x_i), a), W(x_{i+2}), \ldots, W(x_t)\}.$$

By Definitions 13 and 20, $W(x_i a) = \text{Ref}(W(x_i), a)\}$;

c) For $i = 0$, if the letter a appears at a location *index* in p such that *index* is smaller than $Rm(x_1)$, this is equivalent to $Wm(a) > W(x_1)$ by Defintion 20, then $\varphi(C, a) = \{a, x_2, \ldots, x_t\}$ and $\delta(W(C), a) = \{Wm(a), W(x_2), \ldots, W(x_t)\}$. By Definition 13, $W(a) = Wm(a)$;
By (a), (b), (c) above, it follows that $\delta(W(C), a) = W(\varphi(C, a))$. The proof is complete. ∎

Theorem 25. *Let p be a string of length m on the alphabet Σ, $C \in Config(p)$ and $s \in \Sigma^*$. Then $\delta(W(C), s) = W(\varphi(C, s))$, where δ and φ are given as in Definitions 22 and 9, respectively.*

Proof. Consider $s = \epsilon$, by Definitions 22 and 9, $\delta(W(C), s) = W(\varphi(C, s)) = W(C)$. Conversely, consider $s \neq \epsilon$, then suppose $s = s[1..n]$. Now, we prove $\delta(W(C), s) = W(\varphi(C, s))$ using mathematical induction.

Case $n = 1$, by Lemma 24, $\delta(W(C), s[1]) = W(\varphi(C, s[1]))$.

Suppose $\delta(W(C), s) = W(\varphi(C, s))$ is true for some $n = k \geq 1$, that is $\delta(W(C), s[1..k]) = W(\varphi(C, s[1..k]))$.

We prove that $\delta(W(C), s) = W(\varphi(C, s))$ is true for $n = k + 1$. We have $\delta(W(C), s) = \delta(W(C), s[1..k{+}1]) = \delta(\delta(W(C), s[1..k]), s[k{+}1]) = \delta(W(\varphi(C, s[1..k])), s[k{+}1])$ by induction hypothesis. By Lemma 24,

$$\delta(W(\varphi(C, s[1..k])), s[k{+}1]) = W(\varphi(\varphi(C, s[1..k]), s[k{+}1])) = W(\varphi(C, s[1..k{+}1])) = W(\varphi(C, s)).$$

Next, based on Definition 22 and Theorem 25, we propose two automata models to design two corresponding algorithms to solve the Problem 2.

Definition 26. Let p be a string of length m, a be a letter of p and all locations of a in p be $j_1, j_2, \ldots, j_t, 1 \leq j_1 < j_2 < \ldots < j_t \leq m$. Then the weight of a in p, denoted by $W(a)$, is determined by the formula $W(a) = (W^{j_1}(a), W^{j_2}(a), \ldots, W^{j_t}(a))$.

Example 27. Let $p = abcadbad$. Then $W(a) = (8, 5, 2)$.

Definition 28. Let p be a string and *Step* be a positive integer constant, $1 \leq Step \leq |p|$. For $1 \leq i \leq \left\lceil \frac{|p|}{Step} \right\rceil$, the layer i is a set of positive integers, denoted by t_i, is determined by the formula $t_i = \{w | w \in 1..|p|, \left\lceil \frac{w}{Step} \right\rceil = i\}$.

Let a is a letter of p and $W(a) = (w_1, w_2, \ldots, w_t), 1 \leq t \leq m$. The notation $TW(a)$, which is determined by the formula $TW(a) = (tw_1, tw_2, \ldots, tw_t)$, shows that the weight w_i belongs to the layer tw_i, where $tw_i = \left\lceil \frac{w_i}{Step} \right\rceil$ for $1 \leq i \leq t$.

Example 29. Let $p = abcadbad$ and $Step = 3$. Then $t_1 = \{1, 2, 3\},\ t_2 = \{4, 5, 6\},\ t_3 = \{7, 8\},\ W(a) = (8, 5, 2),\ TW(a) = (3, 2, 1)$.

Let $W \in \text{WConfig}(p)$. For $1 \le i \le \left\lceil \frac{|p|}{Step} \right\rceil$, the notation $Tq(i)$ is the location of the element in W with the greatest value among the elements of W in the layer i, by convention if the layer i does not have any elements of W, then the $Tq(i)$ is considered to equal 0. Set $Tq(W) = (Tq(\left\lceil \frac{|p|}{Step} \right\rceil), Tq(\left\lceil \frac{|p|}{Step} \right\rceil - 1), \ldots, Tq(1))$. If $\forall 1 \le i \le \left\lceil \frac{|p|}{Step} \right\rceil$, $Tq(i) = 0$, then denote $Tq(W) = 0$.

Example 30. Let $p = abcadbad$ and $C = \{c, ca, cba, dbad\}$. Then C is a configuration of p, $W = W(C) = \{6, 5, 2, 1\}$, $Tq(1) = 3,\ Tq(2) = 1,\ Tq(3) = 0$. Thus $Tq(W) = (0, 1, 3)$.

Let w is a value in the set $\{1, 2, \ldots, p\}$, the notation $t(w)$ shows that the layer consists of w and is determined by the formula $t(w) = \left\lceil \frac{w}{Step} \right\rceil$.

Example 31. Let $|p| = 8$, $Step = 3$ and $w = 8$. Then $t(w) = 3$.

Definition 32. Let p be a string of length m on the alphabet Σ, $W \in \text{WConfig}(p)$ and $a \in \Sigma$. Then a state transition function δ_{Step} on $\text{WConfig}(p) \times \Sigma$, $\delta_{Step} : \text{WConfig}(p) \times \Sigma \to \text{WConfig}(p)$, is defined as follows.

1. If $a \notin p$, then $\delta_{Step}(W, a) = W$.

2. If $a \in p$ and suppose $W(a) = (a_1, a_2, \ldots, a_{t'})$, $1 \le t' \le m$ and $TW(a) = (ta_1, ta_2, \ldots, ta_{t'})$, then

a) $\delta_{Step}(W_0, a) = \{a_1\}$. Note that $Tq(W_0) = 0$. Update $Tq(ta_1) = 1$;

b) Set $W' = \delta_{Step}(W, a)$. Suppose $W = \{w_1, w_2, \ldots, w_t\}$ for $1 \le t \le m$ and $Tq(W)$ corresponding to W. Then W' is determined by the following sequential algorithm:

$temp = |p| + 1$; $j = 1$; (3.1)
While ($a_j < temp$ and $j \le t$)
{
  $i = Tq(ta_j)$;
  If ($i \ne 0$)
  {
    Case ($w_t > a_j$): $\{w_{t+1} = a_j$; Break;$\}$ (3.2)
    Case ($w_i < a_j$): $\{temp = w_i$; $w_i = a_j$;$\}$ (3.3)
    Case ($w_t < a_j < w_i$):
    {
      $i_1 = i + 1$; While ($w_{i_1} > a_j$) $i_1 + +$;
      If ($w_{i_1} < a_j$)
      {
        $temp = w_{i_1}$; $w_{i_1} = a_j$; (3.4)

If $(t(temp) \neq t(w_i))$
If $(i_1 == t$ or $t(temp) \neq t(w_{i_1+1}))$ $Tq(t(temp)) = 0$;
Else $Tq(t(temp)) = i_1 + 1$;
}
}
} Else { If $(w_t > a_j)$ $\{w_{t+1} = a_j$; $Tq(ta_j) = t + 1$; Break;} (3.5)
$i_1 = ta_j - 1$; While $(Tq(i_1) == 0)$ $i_1 - -$;
$temp = w_{Tq(i_1)}$; $w_{Tq(i_1)} = a_j$; (3.6)
$Tq(a_j) = Tq(i_1)$;
If $(Tq(i_1) == t$ or $t(temp) \neq t(w_{Tq(i_1)+1}))$ $Tq(i_1) = 0$;
Else $Tq(i_1) = Tq(i_1) + 1$;
}
$j + +$; (3.7)
}
$W' = W$;

3. To accept an input string, the state transition function δ_{Step} is extended as follows:

$$\delta_{Step} : \text{WConfig}(p) \times \Sigma^* \to \text{WConfig}(p)$$

such that $\forall W \in \text{WConfig}(p)$, $\forall u \in \Sigma^*$, $\forall a \in \Sigma, \delta_{Step}(W, au) = \delta_{Step}(\delta_{Step}(W, a), u)$ and $\delta_{Step}(W, \epsilon) = W$.

Example 33. Let $p = abcadbad$, $Step = 3$ and $C = \{a, ab, aba, cadb\}$. Then C is a configuration of p, $W = W(C) = \{8, 7, 5, 3\}$, $Tq(W) = (1, 3, 4)$ and $W(d) = (4, 1)$. Thus $W' = \delta_{Step}(W, d) = \{8, 7, 5, 4, 1\}$ and $Tq(W') = (1, 3, 5)$.

Proposition 34. *Let p be a string of length m on the alphabet Σ, $W \in WConfig(p)$ and $a \in \Sigma$. Then $\delta_{Step}(W, a) = \delta(W, a)$, where δ and δ_{Step} are given as in Definitions 22 and 32, respectively.*

Proof. Case $a \notin p$, then $\delta_{Step}(W, a) = \delta(W, a) = W$ by Definitions 22 and 32.
Case $a \in p$, then $\delta_{\text{Step}}(W_0, a) = \{a_1\} = \{Wm(a)\} = \delta(W_0, a)$ by Remark 18, Definitions 22, 26 and 32.
Case $a \in p$, then by Definition 32, W is only and always updated in the following cases:
a) $w_t > a_j$: W is updated by Statements (3.2) or (3.5).
b) $w_{i+1} < a_j < w_i$ for $1 \leq i \leq t - 1$: W is updated by Statements (3.4) or (3.6).
c) $w_1 < a_1$: W is updated by Statements(3.3) or (3.6).
By Defintions 20 and 26, Remark 18, Statements (3.1) and (3.7), we have:
The case (a) is equivalent to $\text{Ref}(w_t, a) \neq 0$ and $a_j = \text{Ref}(w_t, a)$.
The case (b) is equivalent to $\text{Ref}(w_i, a) > w_{i+1}$ and $a_j = \text{Ref}(w_i, a)$.
The case (c) is equivalent to $Wm(a) > w_1$ and and $a_1 = Wm(a)$.
Furthermore, by the definitions of δ and δ_{Step} as in Definitions 22 and 32, then $\delta_{Step}(W, a) = \delta(W, a)$. We complete the proof. ∎

Theorem 35. *Let p and s be two strings of lengths m and n over the alphabet $\Sigma, m \leq n$. Let c be a positive integer constant, $1 \leq c \leq m$ and $\mathcal{A}_p^{Sc} = (\Sigma, Q, q_0, \delta_{Step}, F)$ corresponding to p be an automaton over the alphabet Σ, where*

- *The set of states $Q = WConfig(p)$,*
- *The initial state $q_0 = W_0$,*
- *The transition function δ_{Step} is given as in Definition 32,*
- *The set of final states $F = \{W_f | W_f \in WConfig(p), |W_f| = c \text{ or } W_f = \delta_{Step}(W_0, s)\}$.*

Suppose W_f is a final state. Then there exists a substring u of s such that $lcs(p, u) = |W_f|$.

Proof. Consider the final state of the automaton $\mathcal{A}_p^{Sc}$ of the form $W_f = \delta_{Step}(W_0, s)$, then by Definition 15, Proposition 34 and Theorem 25, $W_f = \delta_{Step}(W_0, s) = \delta_{\text{Step}}(W(C_0), s) = \delta(W(C_0), s) = W(\varphi(C_0, s)) = W(C_f)$, where $C_f = \varphi(C_0, s)$ is the final state of the automaton $\mathcal{A}_p^c$ defined as in Theorem 12, then $u = s$. Otherwise, the final state of the automaton $\mathcal{A}_p^{Sc}$ of the form $W_f \in \text{WConfig}(p), |W_f| = c$, then $\exists u$ is a prefix of s such that $W_f = \delta_{\text{Step}}(W_0, u)$. Similarly, we have $W_f = W(\varphi(C_0, u))$. Set $C_f = \varphi(C_0, u)$, by Definition 15 and Theorem 12, C_f is a final state of configuration of the automaton $\mathcal{A}_p^c$ defined as in Theorem 12 and $C_f = \{x_1, x_2, \ldots, x_c\}$. Suppose $C_f = \{x_1, x_2, \ldots, x_t\}, 1 \leq t \leq m$, then there exists a substring u of s such that a $LCS(p, u) = x_t$ by Theorem 12, thus $lcs(p, u) = |x_t| = t$ by Definition 8. On the other hand, as the proof above, we always have $W_f = W(C_f)$, then by Definition 15, $|W_f| = t$. Therefore $\exists u, lcs(p, u) = |W_f|$. The proof is complete. ∎

Now an application of Theorem 35 with $c = |p|$, we construct a sequential algorithm for solving the Problem 2, as follows.

Algorithm 1 (the sequential algorithm):

Input: Two strings p and s, $|p| \leq |s|$, value of *Step*.

Output: The $lcs(p, s)$.

```
q = W_0; // Set up the initial state of the automaton A_p^{Sc}.
Tq(q) = 0; // Initialize Tq(q).
For i = 1 to |s| Do
{
  q = δ_Step(q, s[i]);
  If (|q| = |p|) Break;
}
lcs(p, s) = |q|;
```

Remark 36. From the definition of δ_{Step} as in Definition 32, we can give a few advantages of the Algorithm 1 in practice:

1. The number of letters of s in p is small.
2. m is much smaller than n.
3. The $lcs(p, s)$ is much smaller than m and n.
4. $Step^2 \approx m$.
5. A $LCS(p, s)$ is a prefix of p. It will be even better if every letter in the $LCS(p, s)$ is only appears once in p.

6. The best case of the Algorithm 1 occurs when $s[i] \notin p, \forall i, 1 \leq i \leq n$ or $s[i] \in p, \forall i, 1 \leq i \leq n$ and it holds that one of two statements (3.2) or (3.5) is executed for $j = 1$. In this case, the time complexity of the Algorithm 1 is $O(n)$.

Definition 37. Let p be a string of length m on the alphabet Σ, $W \in \text{WConfig}(p)$ and $a \in \Sigma$. Then a state transition function $\delta_{\|}$ on $\text{WConfig}(p) \times \Sigma$, $\delta_{\|} : \text{WConfig}(p) \times \Sigma \to \text{WConfig}(p)$, is defined as follows.

1. $\delta_{\|}(W, a) = W$ if $a \notin p$.
2. $\delta_{\|}(W_0, a) = \{ Wm(a)\}$ if $a \in p$.
3. Set $W' = \delta_{\|}(W, a)$. Suppose $a \in p$ and $W = \{w_1, w_2, \ldots, w_t\}$ for $1 \leq t \leq m$. Then W' is determined by the following parallel algorithm:

a) Set $W' = W$;

The following statement block is executed in parallel:

b) If $\text{Ref}(w_t, a) \neq 0$, then $w'_{t+1} = \text{Ref}(w_k, a)$;

c) Execute the following statements in parallel for $\forall i \in \{1, 2, \ldots, t-1\}$, if $\text{Ref}(w_i, a) > w_{i+1}$ then $w'_{i+1} = \text{Ref}(w_i, a)$;

d) If $Wm(a) > w_1$, then $w'_1 = Wm(a)$;

4. To accept an input string, the state transition function $\delta_{\|}$ is extended as follows:

$$\delta_{\|} : \text{WConfig}(p) \times \Sigma^* \to \text{WConfig}(p)$$

such that $\forall W \in \text{WConfig}(p), \forall u \in \Sigma^*, \forall a \in \Sigma, \delta_{\|}(W, au) = \delta_{\|}(\delta_{\|}(W, a), u)$ and $\delta_{\|}(W, \epsilon) = W$.

Proposition 38. *Let p be a string of length m on the alphabet Σ, $W \in WConfig(p)$ and $a \in \Sigma$. Then $\delta_{\|}(W, x) = \delta(W, x)$, where δ and $\delta_{\|}$ are given as in Definitions 22 and 37, respectively.*

Proof. This follows immediately from Definitions 22 and 37. ∎

Theorem 39. *Let p and s be two strings of lengths m and n over the alphabet $\Sigma, m \leq n$. Let c be a positive integer constant, $1 \leq c \leq m$ and $\mathcal{A}_p^{Pc} = (\Sigma, Q, q_0, \delta, F)$ corresponding to p be an automaton over the alphabet Σ, where*

- *The set of states $Q = WConfig(p)$,*
- *The initial state $q_0 = W_0$,*
- *The transition function $\delta_{\|}$ is given as in Definition 37.*
- *The set of final states $F = \{W_f | W_f \in WConfig(p), |W_f| = c \text{ or } W_f = \delta_{\|}(W_0, s)\}$.*

Suppose W_f is a final state. Then there exists a substring u of s such that $lcs(p, u) = |W_f|$.

Proof. This follows immediately from Proposition 38 and Theorem 35. ∎

Based on Theorem 39 with $c = |p|$, we construct a parallel algorithm for solving the Problem 2, as follows.

Algorithm 2 (the parallel algorithm):

Input: Two strings p and s, $|p| \leq |s|$.
Output: The $lcs(p, s)$.

```
q = W_0; // Set up the initial state of the automaton A_p^{Pc}.
For i = 1 to |s| Do
{
  q = δ_||(q, s[i]);                                   (3.8)
  If (|q| = |p|) Break;
}
lcs(p, s) = |q|;
```

Proposition 40. *Let p and s be two strings of lengths m and n over the alphabet $\Sigma, m \leq n$. Suppose the Algorithm 2 uses k processors ($k \leq m$), where k is an upper estimate of the length of a longest common subsequence of the two strings. Then the time complexity of the Algorithm 2 is $O(n)$ in the worst case.*

Proof. By the definition of $\delta_{\|}$ as in Definition 37, at each step of changing the state of the automaton $\mathcal{A}_p^{Pc}$ from the initial state q_0 to an arbitrary final state, the state transition function $\delta_{\|}$ does not use more than $lcs(p, s)$ processors. Since $lcs(p, s) \leq k$, $\delta_{\|}$ is always executed in parallel. Thus, by the definition of $\delta_{\|}$ as in Definition 37, the statement (3.8) takes $O(1)$ time in the worst case. It follows that the time complexity of the Algorithm 2 is $O(n)$ in the worst case. ∎

Remark 41. By Definitions 22 and 37, Propositions 38 and 40, we point the way to determine the running time of the Algorithm 2 if it uses k processors, and give the effective feature of the Algorithm 2 in practice:

1. Assume that the Algorithm 2 runs on a computer with k processors. Then the running time of the Algorithm 2 to compute the $lcs(p, s)$, denoted by T_p, is determined by the formula $T_p = \dfrac{T_s}{|I| + 1} sp$, where T_s is the running time of the Algorithm designed as the Algorithm 1, whose the state transition funtion is defined in Definition 22, to compute the $lcs(p, s)$, sp is the number of letters of s in p, $I = |q_0| + |q_1| + \ldots + |q_{sp-1}|$, where q_i is the state of the automaton with the state transition function determined as in Definition 22 for $0 \leq i \leq sp - 1$.

2. As with the Algorithm 1 if sp is small, then T_p is also small. Suppose that s is a string on the alphabet Σ with a uniform distribution of letters, then sp depends on the probability P that an arbitrary letter of s belongs to p, where $P = \dfrac{m}{|\Sigma|}$. Thus, if Σ is large, then P is small, hence sp is small. So, both algorithms have the advantage of alphabets of the large size.

4 EXPERIMENTAL RESULTS

Let p and s be two strings of lengths m and n over the alphabet Σ. For the $lcs(p, s)$ computation time, in this section we carried out a number of experiments to compare the two proposed algorithms with the Algorithm WF. We used the C# programming language compiled by Microsoft Visual Studio 2010 to implement all algorithms. Our experiments were ran in 64-bit Operating System (Win 7), Intel Core I3, 2.20GHz, 4 GB RAM.

We used the following test data:

• The size of the alphabet Σ is 256.

• Two fixed strings s of lengths 50666 and 102398 with a uniform distribution of letters.

• For each fixed string s, we generate randomly sets of 50 strings p of length m, for m ranging over the values 50, 100, 200, 300, 400, 500, 600, 700, 700, 800, 900, 1000, 2000, 3000, 4000, 5000.

• For each set of strings p, the mean over the running times of the 50 runs is reported in a table corresponding to a certain length of the string s.

Experimental results are shown in two following tables. Each table corresponds to a length of the string s. Denote the running time of the Algorithm WF, the Algorithm 1, the Algorithm 2 and the Algorithm 2 based on the assumption in Remark 41 by T, T_1, T_2 and T_p given as in Remark 41, respectively.

Table 3. The comparisons of the $lcs(p, s)$ computation time for $n = 50666$

m	Algorithm WF T	Algorithm 1		Algorithm 2		T_p		
		T_1	$\frac{T}{T_1}$	T_2	$\frac{T}{T_2}$	T_p	$\frac{T}{T_p}$	$\frac{T}{T_p*m}$
50	0.301997	0.005420	55.7	0.144148	2.1	0.000644	468.9	9.4
100	0.607775	0.009641	63	0.361601	1.7	0.001010	602	6
200	1.236571	0.020701	59.7	0.705160	1.8	0.001580	782.8	3.9
300	1.844046	0.027322	67.5	0.998977	1.8	0.002002	921.1	3.1
400	2.608229	0.035822	72.8	1.192508	2.2	0.002279	1144.5	2.9
500	3.250566	0.045763	71	1.410861	2.3	0.002537	1281.4	2.6
600	3.882162	0.053663	72.3	1.502186	2.6	0.002663	1457.6	2.4
700	4.510698	0.062184	72.5	1.652055	2.7	0.002835	1591.3	2.3
800	5.187317	0.070224	73.9	1.721158	3	0.002871	1806.9	2.3
900	5.788851	0.079725	72.6	1.821924	3.2	0.002906	1992	2.2
1000	6.429848	0.091285	70.4	1.870267	3.4	0.002954	2176.3	2.2
2000	12.794312	0.190351	67.2	2.360195	5.4	0.003164	4044.1	2
3000	19.076211	0.295797	64.5	2.718515	7	0.003244	5880.9	2
4000	25.349450	0.407383	62.2	2.969610	8.5	0.003370	7522.9	1.9
5000	31.522143	0.503049	62.7	3.198803	9.9	0.003457	9119.6	1.8

Table 4. The comparisons of the $lcs(p, s)$ computation time for $n = 102398$

m	Algorithm WF	Algorithm 1		Algorithm 2		T_p		
	T	T_1	$\frac{T}{T_1}$	T_2	$\frac{T}{T_2}$	T_p	$\frac{T}{T_p}$	$\frac{T}{T_p*m}$
50	0.644657	0.011221	57.5	0.395683	1.6	0.001109	581.2	11.6
100	1.345677	0.022722	59.2	0.905212	1.5	0.001969	683.6	6.8
200	2.786899	0.039542	70.5	1.415801	2	0.002562	1087.6	5.4
300	4.074673	0.053423	76.3	1.849586	2.2	0.002969	1372.2	4.6
400	5.436751	0.078685	69.1	2.688234	2	0.004213	1290.5	3.2
500	6.795429	0.094485	71.9	2.865064	2.4	0.004322	1572.3	3.1
600	8.153206	0.132428	61.6	3.502480	2.3	0.005086	1603.1	2.7
700	9.502244	0.141588	67.1	3.741414	2.5	0.005275	1801.5	2.6
800	10.825719	0.164149	66	3.781196	2.9	0.005229	2070.5	2.6
900	12.136634	0.179110	67.8	4.024410	3	0.005400	2247.6	2.5
1000	13.460410	0.215552	62.4	4.437774	3	0.005795	2322.6	2.3
2000	26.620703	0.405343	65.7	5.736688	4.6	0.006371	4178.4	2.1
3000	39.309348	0.733762	53.6	6.270719	6.3	0.006559	5992.9	2
4000	52.526324	0.808566	65	6.820750	7.7	0.006734	7800.3	2
5000	65.219030	1.211189	53.8	7.395623	8.8	0.006909	9439.9	1.9

Experimental results show the outstanding advantages of the two algorithms proposed in the practice. If calculate the average of two above tables, we see that the Algorithm 1 and Algorithm 2 based on T_p time are about 65.85 and $3.41m$ times faster than the Algorithm WF, respectively.

Note that the Algorithm 2 based on T_2 time only illustrates the possibility of parallel installation.

5 CONCLUSIONS

In this paper, we have introduced the mathematical basis for the development of the automata technique for computing the $lcs(p, s)$ based on Knapsack Shaking approach to finding a $LCS(p, s)$ [6]. By using automata proposed, we presented two algorithms to compute the $lcs(p, s)$. The parallel algorithm takes $O(n)$ time in the worst case if it uses k processors, where k is an upper estimate of the length of a longest common subsequence of the two strings p and s. Experimental results also show the efficiency of our approach in designing algorithms for computing the $lcs(p, s)$.

The structures of the automata proposed are only based on the preprocessing of the string p. Thus, our algorithms will have many advantages for the approximate pattern matching between one pattern and one very large set of the texts.

The $lcs(p, s)$ is always reflected and updated at every location being scanned in the string s, then our two algorithms can be applied to secure data environment. These applications will be introduced in the next works.

ACKNOWLEDGMENT

The author is greatly indebted to Late Assoc. Prof. Phan Trung Huy and Assoc. Prof. Phan Thi Ha Duong for their valuable suggestions and comments.

This work was partially funded by the Vietnam National Foundation for Science and Technology Development (NAFOSTED) under the grant number 101.99-2016.16.

REFERENCES

[1] A. V. Aho, D. S. Hirschberg, J. D. Ullman, "Bounds on the complexity of the longest common subsequence problem," *Journal of the Association for Computing Machinery*, vol. 23, no. 1, pp. 1–12, 1976.

[2] A. Begum, "A greedy approach for computing longest common subsequences", *Journal of Prime Research in Mathematics*, vol. 4, pp. 165–170, 2008.

[3] V. Chvatal, D. A. Klarner, D. E. Knuth, "Selected combinatorial research problems", *Stan-CS-TR-72-292*, pp. 26, 1972.

[4] A. Dhraief, R. Issaoui, A. Belghith, "Parallel computing the longest common subsequence (LCS) on GPUs: Efficiency and language suitability," *Proceedings of the 1st International Conference on Advanced Communications and Computation*, Spain, October 23-28, 2011, pp. 143-148.

[5] D. S. Hirschberg, "A linear space algorithm for computing maximal common subsequences", *Comm. ACM*, vol. 18, no. 6, pp. 341–343, 1975.

[6] P. T. Huy, N. Q. Khang, "A new algorithm for LCS problem", *Proceedings of the 6th Vietnam Conference of Mathematics*, Hue, September 7-10, 2002, pp. 145–157.

[7] C. S. Iliopoulos, M. S. Rahman, "A new efficient algorithm for computing the longest common subsequence", *Theory Comput Syst*, vol. 45, pp. 355-371, 2009.

[8] Indu, Prena, "Comparative study of different longest common subsequence algorithms", *International Journal of Recent Research Aspects*, vol. 3, no. 2, pp. 65–69, 2016.

[9] T. Jiang, M. Li, "On the approximation of shortest common supersequences and longest common subsequences, *SIAM J. Comput*, vol. 24, no. 5, pp. 1122–1139, 1995.

[10] J. V. Leeuwen, "Handbook of theoretical computer science", vol. A, *Elsevier MIT Press*, pp. 290–300, 1990.

[11] D. Maier, "The complexity of some problems on subsequences and supersequences", *Journal of the ACM*, vol. 25, no. 2, pp. 322–336, 1978.

[12] P. H. Paris, N. Abadie, C. Brando, "Linking spatial named entities to the Web of data for geographical analysis of historical texts", *Journal of Map & Geography Libraries*, vol. 13, no. 1, pp. 82–110, 2017.

[13] M. V. Ramakrishnan, S. Eswaran, "A comparative study of various parallel longest common subsequence (LCS) algorithms", *International Journal of Computer Trends and Technology*, vol. 4, no. 2, pp. 183–186, 2013.

[14] R. A. Wagner, M. J. Fischer, "The string-to-string correction problem", *J. ACM*, vol. 21, no. 1, pp. 168–173, 1974.

[15] X. Xu, L. Chen, Y. Pan, P. He, "Fast parallel algorithms for the longest common subsequence problem using an optical bus", *Computational Science and Its Applications, ICCSA 2005, Proceedings, Part III*, Singapore, May 9-12, 2005, pp. 338–348.

[16] J. Yang, Y. Xu, Y. Shang, "An efficient parallel algorithm for longest common subsequence problem on GPUs", *Proceedings of the World Congress on Engineering,* vol. 1, London, June 30 - July 2, 2010, pp. 499–504.

Received on November 12, 2018
Revised on February 14, 2019