A NEW ALGORITM FOR THE LONGEST COMMON SUBSEQUENCE PROBLEM

Vahagn Minasyan

***Abstract**: This paper discusses the problem of determining the longest common subsequence (LCS) of two sequences. Here we view this problem in the background of the well known algorithm for the longest increasing subsequence (LIS). This new approach leads us to a new online algorithm which runs in $O(d\ log\ n)$ time and in $O(n)$ space where n is the length of the input sequences and d is the number of minimal matches between them. Using an advanced technique of van Emde Boas trees the time complexity bound can be reduced to $O(d\ log\ log\ n)$ preserving the space bound of $O(n)$.*

***Keywords**: longest common subsequence, longest increasing subsequence, online algorithm.*

***ACM Classification Keywords**:G.2.1 Discrete mathematics: Combinatorics*

Introduction

Let $A = a_1 \cdots a_i \cdots a_m$and $B = b_1 \cdots b_j \cdots b_n$, $1 \leq m \leq n$, be two sequences over some alphabet Σ of size s, $s \geq 1$. A sequence $C = c_1 \cdots c_k \cdots c_l$, $1 \leq l$, over Σ is called a *subsequence* of A, if C can be obtained from A by deleting some of its elements, that is if exists a set of indices$\{i_1, \ldots, i_k, \ldots, i_l\}$ such that $1 \leq i_1 < \cdots < i_k < \cdots < i_l \leq m$ and $c_k = a_{i_k}$ for $1 \leq k \leq l$. C is said to be a *common subsequence* of A and B, if it is a subsequence of both sequences A and B;C is said to be a *longest common subsequence (LCS)* of A and B, if it has the maximum length among all common subsequences of A and B; that length is called the *LCS length* of A and B. In general the longest common subsequence is not unique.

The *Longest Common Subsequence Problem (LCS Problem)* is to determine a LCS of $A$and B. Oftenthe problem of determining theLCS length is also referred to as LCS Problem. This is due to the fact that most of algorithms intended to find the LCS length can easily be modified to determine a LCS [Bergroth, 2000]. In this paper we will concentrate on determining the LCS length rather thandetermining an actual LCS.The first known solution of the LCS Problem is based on dynamic programming [Cormen, 2009].For $1 \leq i \leq m$ and $1 \leq j \leq n$ denote by $l_{i,j}$ the LCS length of sequences $a_1 \cdots a_i$and $b_1 \cdots b_j$; thus $l_{m,n}$ is the LCS length of A and B. Note that thefollowing recursion holds for $l_{i,j}$:

$$l_{i,j} = \begin{cases} 0 \text{ if } i = 0 \text{ or } j = 0 \\ l_{i-1,j-1} + 1 \text{ if } a_i = b_j \\ \max\{l_{i-1,j}, l_{i,j-1}\} \text{ if } a_i \neq b_j \end{cases} \quad (1)$$

Based on this relation it is easy to construct an algorithm which fills an array of size $m \times n$, where (i,j)-th cell contains the value of $l_{i,j}$. As it follows form (1) such algorithm has to fill the rest of array before obtaining the value of (m,n)-th cell, so it will determine the LCS length of sequences A and Bin $\Theta(mn)$ time and $\Theta(mn)$ space ($\Theta(1)$ time for filling each cell and $\Theta(1)$ space for holding each cell).A simple trick can be used to make this algorithm require only $\Theta(m+n)$ space to obtain the value of the (m,n)-th cell [Cormen, 2009]. Here we give some definitions which will be used later in the paper. For $1 \leq i \leq m$ and $1 \leq j \leq n$ the pair (i,j) is called *matching* between sequences A and B if $a_i = b_j$; it is called *minimal (or dominant) matching* if for every other matching (i',j') such that $l_{i,j} = l_{i',j'}$ it holds $i' > i$ and $j' \leq j$ or $i' \leq i$ and $j' > j$.Note that if m' and n' are two integers such that $m \leq m'$and $n \leq n'$, then theLCS Problem for two sequences of size m and n is asymptotically not harder than the LCS Problem for two sequences of size m' and n'. Indeed, given two sequences of size m and n and an algorithm which solves the LCS Problem for two sequences of size m' and n', we can lengthen the given sequences (by appending to themsymbols which don't occur in the initial sequences) up to size m' and n' respectively and pass the resulting two sequences to the given algorithm. It is easy to see that such algorithm will solve the LCS Problem for two sequences of size m and n in asymptotically the same time and space bounds as the given algorithm solves the LCS Problem for two sequences of size m' and n'. This means that each lower bound for the LCS Problem for two sequences of size m andeachupper bound for the LCS Problem for two sequences of size n are respectively lower and upper bounds for the LCS Problem for two sequences of size m and n (recall that $m \leq n$). At [Aho, 1976] the LCS Problem is examined using the decision tree model of computation where the decision tree vertices represent "equal-unequal" comparisons. There it is shown that each algorithm solving the LCS Problem and fitting this model has time complexity lower bound of $\Omega(ms)$, where s is the number of distinct symbols occurring in the sequences (i.e. the alphabet size). This means that the LCS Problem with unrestricted size of the alphabet has time complexity lower bound of $\Omega(mn)$, as such LCS Problem can be viewed as an LCS Problem with restricted alphabet of size $(m+n)$.In practice the underlying encoding scheme for the symbols of the alphabet implies a topological order between them. Algorithms which take into account this fact don't fit the decision tree model with "equal-unequal" comparisons examined at [Aho, 1976]. At [Masek, 1980] it is presented an algorithm which applies the "Four Russians" trick to the dynamic programming approach, thusit doesn't fit the model examined at [Aho, 1976] and has time complexitybound of$O(mn/\log m)$.This bound isasymptotically the best known for general case LCS Problem[Cormen, 2009].

Previous Results

Lot of algorithms have been developed for the LCS Problem that, although not improving the time complexity bound $O(mn)$, exhibit much better performance for some classes of sequences A and B [Bergroth, 2000]. Consider the special case when the alphabet Σ consists of first n integers, i.e. $\Sigma = \{1, \dots, j, \dots, n\}$, and the sequences A and B are two permutations of Σ. It is easy to check that this case can be reduced to the case where B is the identical permutation (by replacing b_j by j for $1 \leq j \leq n$ in both sequences A and B we will get two sequences which are equivalent to the initial ones with respect to the LCS Problem). In this case each LCS of A and B is an increasing sequence of some of first n integers and each such sequence is a LCS of $A$and B. Thusin the case when A and $B$are permutations the LCS Problem is reduced to the problem of determining a longest increasing subsequence of permutation B. The *Longest Increasing Subsequence (LIS) Problem* is to determine a non decreasing subsequence of maximum length in the given sequence of integers. The LIS Problemcan be solved in $O(n \log n)$time [Fredman, 1975], and using advanced data structures like van Emde Boas trees [Cormen, 2009]thistime bound can be reduced to $O(n \log\log n)$. Thus these bounds apply to the LCS Problem in the case of permutations.Also there are many algorithms for the general case LCS Problem which except$m$ and $n$are also sensitive for other parameters like the LCS length, the alphabet size, the number of matches and the number of minimal matches. A survey on such algorithms is given at [Bergroth, 2000].The table below gives a brief remark of some of known algorithms for the LCS Problem. There l denotes the LCS length, s denotes the alphabet size, r denotes the number of all matches and d denotes the number of minimal matches. It is known [Baeza-Yates, 1999] that for two random sequences of length n the expected LCS length is $O(n)$ and the expected number of minimal matches is $O(n^2)$ [Tronicek, 2002]. This means that (except the 5th) none of the algorithms mentioned in the table hastime complexity upper bound less than$O(mn)$not only in the worst case but also in the average case.

All these algorithms are developed in the background of building the $m \times n$ array mentioned in the dynamic programming approach, and they purport to perform fewer operations in order to obtain the (m, n)-th cell of that array. In this paper we view the LCS Problem inanother background, namely the background of the classical algorithm for the LIS Problem described at [Fredman, 1975]. For sure each term we deal with in this background has its direct analogue in the background of the $m \times n$ array; however our approach can be justified by the fact that it leadsus to simpler constructions and an $O(d \log n)$ algorithm for the LCS Problem which can be reduced to $O(d \log\log n)$ if using van Emde Boas trees (details are in the next section). Initially algorithms from 10th to 16th require $O(ns)$ space, but at [Apostolico, 1987] a trick is introduced which can be used to reduce the space complexity to $O(n)$, however in this case the time complexity bounds increase by a multiplicative factor of $\Theta(\log s)$. The 9th algorithm requires $O(ns)$ space but that trick cannot be used to reduce this space complexity bound[Apostolico, 1987]. Recall that d is the number of minimal matches. It can be checked that $d \leq l(m-l)$ [Rick, 1994] and it is known that in average it holds $d = \Theta(mn)$ [Tronicek, 2002]. This means the 9th,10th and 14th algorithms mentioned in the table above have better time complexity bounds than the others mentioned there. The algorithm we present here has better time complexity bound than 10th and 14th in case when $s = \omega(\log n)$ (or $s = \omega(\log \log n)$ if the van Emde Boas trees are used), and it has better space complexity bound than 9th in cases when $s = \omega(1)$ (see [Cormen, 2009] for the ω-notation). Roughly speaking the algorithm we present here has better time and space complexity bounds than the ones mention in the table above when the alphabet size if relevantly larger. We present the algorithm in the next section.

No.	Year	Authors	Time Complexity	Ref.
1	1974	Wagner, Fischer	$O(mn)$	[Cormen, 2009]
2	1977	Hunt, Szymansky	$O(m + r \log l)$	[Hunt, 1977]
3	1977	Hirschberg	$O(ln)$	[Hirschberg, 1977]
4	1977	Hirschberg	$O(l(m-l) \log n)$	[Hirschberg, 1977]
5	1980	Masek, Paterson	$O(mn/\log m)$	[Masek, 1980]
6	1982	Nakatsu et al.	$O\big(n(m-l)\big)$	[Nakatsu, 1982]
7	1984	Hsu, Du	$O(lm \log(n/l))$	[Hsu, 1984]
8	1986	Myers	$O\big(n(n-l)\big)$	[Myers, 1980]
9	1987	Apostolico, Guerra	$O(m \log n + d \log(2mn/d))$	[Apostolico, 1987]
10	1987	Apostolico, Guerra	$O(lm \log(2n/m))$	[Apostolico, 1987]
11	1990	Chin, Poon	$O(ns + \min\{lm, ds\})$	[Chin, 1990]
12	1990	Wu, Manber, Myers	$O\big(n(m-l)\big)$	[Wu, 1990]
13	1992	Apostolico et al.	$O\big(n(m-l)\big)$	[Apostolico, 1992]
14	1994	Rick	$O(ns + \min\{lm, l(n-l)\})$	[Rick, 1994]
15	1994	Rick	$O(ns + \min\{lm, ds\})$	[Rick, 1994]
16	2002	Goeman, Clausen	$O(ns + \min\{lm, l(n-l)\})$	[Goeman, 2002]

The New Algorithm

First we will discuss the algorithm for the LIS Problem presented at [Fredman, 1975]. That algorithm is an *online algorithm* meaning that it sequentially handles the elements of the input sequence and determines the LIS length of the sequence handled so far.Online algorithms have advantage that they can run on dynamically changing input data. For instanceunlike the Selection Sort, the Insertion Sort algorithm can maintain the sorted list upon the appendingof the next element to the input list [Cormen, 2009]. Thus such algorithms are defined as update procedures which are to be performed upon the appending of the next element. Now back to the LIS Problem.Let $A = a_1 \cdots a_i \cdots a_m$ be a sequence of integers and let x be an integer which is being appended to A. We will describe an online algorithm which determines the LIS length of A' which is A appended by x. Denote by l the LIS length of A and by l' the LIS lengthof A'. Note that $l' = l$or $l' = l + 1$. For $1 \leq k \leq l$ there are increasing subsequences of length $k$in A. Let $x_k$be the minimumof their last elements. It is easy to check that

$$x_1 \leq \cdots \leq x_k \leq \cdots \leq x_l \tag{2}$$

We denote by x'_k the analogue of $x_k$inA': for $1 \leq k \leq l'$ let x'_k denote the minimum of the last elements of increasing subsequences of length k of A'. In order to obtain an online algorithm for the LIS Problem we will describe how to determine values $(x'_k)_{k=1}^{l'}$ based on values $(x_k)_{k=1}^{l}$. Firstly note that $l' = l + 1$ if and only if $x_l \leq x$, and if so then $x'_{l+1} = x$. It is easy to check that this claim can be generalized for any $1 \leq k \leq l$: let r denote $l + 1$ if $x_l \leq x$ and otherwise let r be the least index such that $x < x_r$. It is easy to check that for $k = r$ it holds $x'_k = x$and otherwise $x'_k = x_k$. Thus we have described a way how to obtain values $(x'_k)_{k=1}^{l'}$ based on values $(x_k)_{k=1}^{l}$. Next the online algorithm for the LIS Problem is described. The algorithm maintains the values $(x_k)_{k=1}^{l}$ in an array $\mathbf{endpoints}$. Upon the appending of the next element x to sequence A the algorithm just searches for the index r mentioned above and updates the value at that index.

LIS-update

Input: the next element $\mathbf{x}$ of sequence A

Output:the LIS length of the sequences$A$ handled so far

Method:

```
1. $\mathbf{if}\ (\ \mathbf{x}\ has\ no\ upper\ bound\ in\ \mathbf{endpoints})\ \mathbf{then\ do}$
2. $append\ (\ \mathbf{endpoints},\ \mathbf{x}\ )$
3. $\mathbf{done\ else\ do}$
4. $\mathbf{endpoints}[upper\_bound\ (\ \mathbf{endpoints},\ \mathbf{x}\ )] = \mathbf{x}$
5. $\mathbf{done}$
6. $\mathbf{output}\ size\ (\ \mathbf{endpoints}\ )$
```

Note that each call of this procedure requires $\Theta(\log l)$ time where l is the LIS length of the sequence handled so far. Thus we have described an online algorithm for the LIS Problem which runs in $\Theta(m \log l)$ time and in $\Theta(l)$ space where m is the length of the sequence handled so far and l is the LIS length of that sequence. Next we will present an online algorithm for the LCS Problem which determines the LCS length of two sequences of length $m$and n, $m \leq n$, in $O(d \log n)$ time where d is the number of minimal matches between the input sequences. As for the LCS Problem there are two input sequences some clarification is needed regarding the notion of online algorithms.By an online algorithm for the LCS Problem we mean an algorithm which can accept the next element of either of the two input sequences and provide the LCS length of the two sequences handled so far. Let $A = a_1 \cdots a_i \cdots a_m$ and $B = b_1 \cdots b_j \cdots b_n$ be two sequences over some alphabet Σ of size s and let $y \in \Sigma$ be a symbol being appendedto B. We will describe an online algorithm which determines the LCS length of $A$and B', where B' is Bappended by y. Denote by l the LCS length of A and B and by l' the LCS length of $A$and B'. Note that $l' = l$or $l' = l + 1$. For $1 \leq k \leq l$ there are subsequences of length $k$common to $A$and B. Let i_k be the minimum index such that there is a subsequence of length $k$common to$A$ and B ending at $i_k$in A.It is easy to check that

$$i_1 < \cdots < i_k < \cdots < i_l \tag{4}$$

Similarly for $1 \leq k \leq l$ we define j_k as the minimum index such that there is a subsequence of length k common to A and B ending at $j_k$in B, and we get

$$j_1 < \cdots < j_k < \cdots < j_l \tag{5}$$

We will call the indices at (4) *thresh indices* or *thresh values* of sequence$A$with respect to B and the indices at (5) thresh indices or thresh values of sequence B with respect to A. Let for $1 \leq k \leq l' i'_k$ be the thresh values of sequence$A$with respect to B' and j'_k be the thresh values of B' with respect to A. In order to obtain an online algorithm for the LCS Problem we will describe how to determine indices $(i'_k)_{k=1}^{l'}$ and $(j'_k)_{k=1}^{l'}$ based on indices $(i_k)_{k=1}^{l}$and $(j_k)_{k=1}^{l}$. Firstly note that $l' = l + 1$ if and only if there is some index r, $l < r \leq m$, such that $x_r = y$, and if so then i'_{l+1} is the minimum of such r-s. It is easy to check that this claim can be generalized for any $1 \leq k \leq l$: if r is the first occurrence of y in A after i_{k-1} and $r < i_k$ then $i'_k = r$ and otherwise $i'_k = i_k$ (see Figure 1).

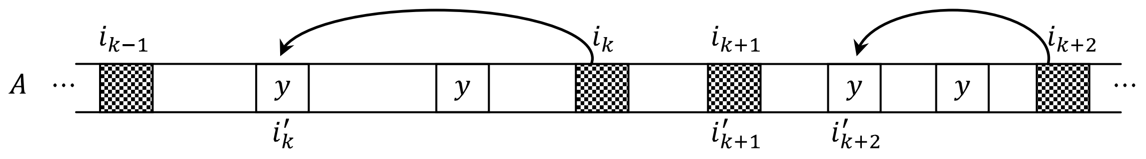

Figure 1

Thus we have described a way how to obtain sequences A and B' and their thresh indices based on sequences A and B and their thresh indices.So during this some thresh values are updated and the others are not.A trivial approach would be to handle all thresh values and update them if they has to be updated, however better would be to handle only those thresh values which has to be updated.Let r be the first occurrence of y in A aftersomei_k. Note that the least thresh value exceeding i_k which has to be updated is the first occurrence of thresh value after r. This means that while searching for the first occurrence of y (after some thresh value) the thresh values can be ignored.Also note that the thresh values of B' with respect to A, i.e. the $(j'_k)_{k=1}^{l'}$, can be obtained easily:there is a new thresh value there if and only if $l' = l + 1$ and if so then $j'_{l+1} = n + 1$. It can be checked that each update of a thresh value corresponds to a minimal match.Next the algorithm is presented. It consists of two update procedures: one for calling upon the appending the next element to sequence A and another upon the appending the next element tosequence B. We will restrict only on the second one as the first one can be obtained just by swapping symbols "A" and "B" in the text of the procedure. The algorithm maintains the sequences$A$ and B in arrays $\mathbf{sequenceA}$and $\mathbf{sequenceB}$ respectively and for each symbol$z$ of alphabet Σ it maintains the set of occurrences of$z$ in A and B in binary search trees $\mathbf{layersA}[z]$ and $\mathbf{layersB}[z]$ respectively.The algorithm also maintains the thresh indices $(i_k)_{k=1}^{l}$ and $(j_k)_{k=1}^{l}$ in binary search trees $\mathbf{threshA}$ and $\mathbf{threshB}$ respectively. Following is the update procedure which is to be called upon the appending the next element to sequence B.The procedure uses two temporary variables $\mathbf{p}$ and $\mathbf{q}$which correspondto the next and previous values of updating thresh indices.

Note that each iteration of the while loop at lines 3-12 updates a thresh value ($\mathbf{p} < \mathbf{q}$ at the end of each iteration) and the operations carried out during each iteration require $\Theta(\log n)$ time as they are performed on binary search trees. Recall that each update of the thresh value corresponds to a minimal match, so we have described an online algorithm for the LCS Problem which runs in $\Theta(d \log n)$ time and in $\Theta(m + n)$ space where$m$ and n are the lengths of the sequences handled so far and d is the number of minimal matches between that sequences. These bounds can be improved if using van Emde Boas trees [Cormen, 2009] instead of binary search trees.van Emde Boas tree is a data structure that for some a priory fixed integer$w$ can store some of first 2^w integers,itsupports operations of insertion deletion and search for the upper bound with worst case time complexity bound of $\Theta(\log\log n)$and it requires $\Theta(2^w)$ space regardless the number of integers stored in it. At [Cormen, 2009] it is shown how this data structure can be modified to require only $\Theta(n)$ space where n is the number of stored elements (there the modified data structure is called y-fast trie). In this case the operations of insertion and deletion do not have worst case time complexity bound of $\Theta(\log\log n)$ but this bound holds for the amortized time complexity. This fits with our needs as we perform $\Theta(d)$ insertions and deletions, thus we conclude that if using these modified van Emde Boas trees then the algorithm presented in this paper will run in $\Theta(d \log\log n)$ time and in $\Theta(n)$ space.

LCS-updateB

Input: the next element $\mathbf{y}$ of sequence$B$

Output:the LCS length of the sequences$A$ and Bhandled so far

Method:

```
1.  $\mathbf{p} = 0$
2.  $\mathbf{q} = 0$
3.  $\mathbf{while}\ (\ \mathbf{true}\ )\ \mathbf{do}$
4.  $\mathbf{if}\ (\ \mathbf{q}\ \mathit{has\ no\ upper\ bound\ in}\ \mathbf{layersA}[\mathbf{y}])\ \mathbf{then\ break}$
5.  $\mathbf{p} = \mathit{upper\_bound}\ (\ \mathbf{layersA}[\mathbf{y}],\ \mathbf{q}\ )$
6.  $\mathit{erase}\ (\ \mathbf{layersA}[\mathbf{sequenceA}[\mathbf{p}]],\ \mathbf{p}\ )$
7.  $\mathbf{if}\ (\ \mathbf{p}\ \mathit{has\ no\ upper\ bound\ in}\ \mathbf{threshA})\ \mathbf{then\ break}$
8.  $\mathbf{q} = \mathit{upper\_bound}\ (\ \mathbf{threshA},\ \mathbf{p}\ )$
9.  $\mathit{insert}\ (\ \mathbf{layersA}[\mathbf{sequenceA}[\mathbf{q}]],\ \mathbf{q}\ )$
10. $\mathit{erase}\ (\ \mathbf{threshA},\ \mathbf{p}\ )$
11. $\mathit{insert}\ (\ \mathbf{threshA},\ \mathbf{q}\ )$
12. $\mathbf{done}$
13. $\mathbf{if}\ (\mathbf{p}\ \mathit{has\ no\ upper\ bound\ in}\ \mathbf{threshA})\ \mathbf{then\ do}$
14. $\mathit{insert}\ (\ \mathbf{threshA},\ \mathbf{p}\ )$
15. $\mathit{insert}\ (\ \mathbf{threshB},\ \mathit{size}\ (\ \mathbf{sequenceB}\ ))$
16. $\mathbf{done\ else\ do}$
17. $\mathit{insert}\ (\ \mathbf{layersB}[\mathbf{y}],\ \mathit{size}\ (\ \mathbf{sequenceB}\ ))$
18. $\mathbf{done}$
19. $\mathit{append}\ (\ \mathbf{sequenceB},\ \mathbf{y}\ )$
20. $\mathbf{output}\ \mathit{size}\ (\ \mathbf{threshA}\ )$
```

Bibliography

[Aho, 1976]A. Aho, C. Hirschberg, J. Ullman. Bounds on the Complexity of the Longest Common Subsequence Problem. Journal of the Association for Computing Machinery, Vol. 23, No. 1, 1976, pp. 1-12.

[Apostolico, 1987]A. Apostolico, G. Guerra. The longest common subsequence problem revisited. Algorithmica 2, 1987, pp. 315-336.

[Apostolico, 1992]. Apostolico, S. Browne, C. Guerra. Fast Linear-Space Computations of Longest Common Subsequences. Theoretical Computer Science, Vol. 92, 1992, pp. 3-17.

[Baeza-Yates, 1999] R. Baeza-Yates, R. Gavalada, G. Navarro, R. Scheihing.Bounding the Expected Length of Longest Common Subsequences and Forestes.Theory of Computing Systems, Vol. 32, 1999, pp. 435-452.

[Bergroth, 2000]L. Bergroth, H. Hakson, T. Ratia.A Survey of Longest Common Subsequence Algorithms.Proceedings of the Seventh International Symposium on String Processing Information Retrieval, 2000, pp. 39-48.

[Chin, 1990]. F. Chin, C. Poon. A Fast Algorithm for Computing Longest Common Subsequences of Small Alphabet Size.Journal of Information Processing, Vol.13, No. 4,1990, pp. 463-469.

[Cormen, 2009] T. Cormen, C. Leiserson, R. Rivest, C. Stein. Introduction to algorithms (Third Edition), pp. 43-65, pp. 390-397, pp.531-561, 2009.

[Fredman, 1975] M. Fredman. On Computing the of Longest Increasing Subsequences. Discrete Mathematics, Vol. 11, No. 1, 1975, pp. 29-35.

[Goeman, 2002]H. Goeman, M. Clausen. A new practical linear space algorithm for the longest common subsequence problem.Kybernetika, Vol. 38, No. 1, 2002, pp. 45-66.

[Hirschberg, 1977]D. Hirschberg.Algorithms for the Longest Common Subsequence Problem.Journal of Association for Computing Machinery, Vol. 24, No. 4, 1977, pp. 664-675.

[Hsu, 1984]W. Hsu, M. Du. New algorithms for the LCS problem. Journal of Computer and Syste Sciences, Vol. 29, 1984, pp. 133-152.

[Hunt, 1977]J. Hunt, T. Szymanski. A fast algorithm for computing longest common subsequences.Communications of Association for Computing Machinery, Vol. 20, No. 5, 1977, pp. 350-353.

[Masek, 1980] W. Masek, M. Paterson. A Faster Algorithm Computing String Edit Distances. Journal of Computer and System Sciences, Vol. 20, No. 1, 1980, pp. 18-31.

[Myers, 1986]E.Myers. An $O(ND)$ Difference Algorithm and its Variations. Algorithmica 1, 1986, pp. 251-266.

[Nakatsu, 1982]N. Nakatsu, Y. Kambayashi, S. Yajima. A Longest Common Subsequence Algorithm Suitable for Similar Text Strings. Acta Informatica Vol. 18, 1982, pp. 171-179.

[Rick, 1994] C. Rick. New Algorithms for the Longest Common Subsequence Problem.Research Report No. 85123-CS, Department of Computer Science, University of Bonn, 1994.

[Tronicek, 2002]Z. Tronicek.Common Subsequence Automatation.Research Report DC-2002-02, Department of Computer Science and Engineering, Czech Technical University, 2002.

[Wu, 1990] S. Wu, U. Manber, G. Myers, W. Miller.An $O(NP)$ Sequence Comparison Algorithm.Information Processing Letters, Vol. 35, 1990, pp. 317-323.

Authors' information

VahagnMinasyan *– Postgraduate student, Yerevan State University, faculty of Informatics and Applied Mathematics, department of Discrete Mathematics and Theoretical Informatics, Armenia, 0025, Yerevan, 1st Alex Manoogian; e-mail: vahagn.minasyan@gmail.com*