The Longest Restricted Common Subsequence Problem[1]

Gabriela Andrejková

Department of Computer Science, College of Science,
P. J. Šafárik University,
Jesenná 5, 041 54 Košice, Slovakia

e-mail: `andrejk@kosice.upjs.sk`

Abstract. An efficient algorithm is presented that solves a Longest Restricted Common Subsequence Problem (RLCS) of two partitioned strings with the restricted using of elements. The above algorithm has an application in solution of the Set-Set Longest Common Subsequence Problem (SSLCS). It is shown the transformation of SSLCS Problem on RLCS Problem.

Key words: Design and analysis of algorithms, longest common subsequence, dynamic data structures.

1 Introduction

The common subsequence problem of two strings is to determine one of the subsequences that can be obtained by deleting zero or more symbols from each of the given strings.

The longest common subsequence problem (*LCS Problem*) of two strings is to determine the common subsequence with the maximal length.

For example, the strings *AGI* is a common subsequence and the string *ALGI* is the longest common subsequence of the strings *ALGORITHM* and *ALLEGATION*.

Algorithms for this problem can be used in chemical and genetic applications and in many problems concerning to the data and to the text processing. Genetic and chemical applications comprise the study of differences between long molecules such as proteins [14]. In the data processing and in the text processing the algorithms are used to determine an equivalence or a similarity of two strings [11] and to compress data when similar texts are being stored [4].

Further applications include the string-to-string correction problem [11] and determining the measure of differences between text files [4]. The length of the longest common subsequence (*LLCS Problem*) can determine the measure of differences (or similarities) of text files.

D. S. Hirschberg [6] presented $O(p \cdot n)$-time and $O(p \cdot (m-p) \cdot log\, n)$-time LCS algorithms, where m, n are the lengths of strings and p is the length of any longest common subsequence.

[1]This research was supported by Slovak Grant Agency for Science VEGA, Project No. 1/4375/97

J. W. Hunt and T. G. Szymanski [10] have presented $O((m+r)\cdot log\, n)$-time and $O(m+r)$-space algorithm, where m, n are lengths of strings and $r = |\{\langle i,j\rangle : a_i = b_j, 1 \le i \le m, 1 \le j \le n\}|$. G. Andrejková, Y. Robert and M. Tchuente [1, 15, 16] have presented systolic systems for LCS Problem with the combined complexity measures - $A\cdot T^2 = O(n^3)$ and $A\cdot P^2 = O(n^2)$, where A, T, P are complexity measures: area, time and period.

D. S. Hirschberg and L. L. Larmore [7] have discussed a generalization of LCS Problem, which is called Set LCS Problem (*SLCS Problem*) of two strings where however the strings are not of the same type. The first string is a sequence of the symbols and the second string is a sequence of subsets over an alphabet Ω. The elements of each subset can be used as an arbitrary permutation of elements in the subset. The longest common subsequence in this case is a sequence of symbols with maximal length. The SLCS Problem has an application to problems in computer driven music [7]. D. S. Hirschberg and L.L. Larmore have presented $O(m\cdot n)$-time and $O(m+n)$-space algorithm, m, n are lengths of strings.

The Set-Set LCS Problem (*SSLCS Problem*) is discussed by D. S. Hirschberg and L. L. Larmore [8] in 1989. In this case both strings are the strings of subsets over an alphabet Ω. In the paper is presented an $O(m\cdot n)$-time algorithm which solves the general SSLCS Problem.

In this paper we present an algorithm for special case of the LCS Problem, it means Longest Restricted Common Subsequence Problem (*LRCS Problem*) and its using to the solution of SSLCS Problem.

2 Basic Definitions

In this section, some basic definitions and results concerning to LRCS Problem and SSLCS Problem are presented.

Let Ω be a finite alphabet, $|\Omega| = s$, $P(\Omega)$ the set of all subsets of Ω, $|P(\Omega)| = 2^s$.

Let $A = a_1 a_2 \ldots a_m, a_i \in \Omega, 1 \le i \le m$ be a string over an alphabet Ω, $|A| = m$ is the length of the string A. A sequence of indices, $\mathrm{h}^{\mathrm{A}} = h_0^A h_1^A h_2^A \ldots h_{k^A}^A, 0 = h_0^A < h_1^A < h_2^A < \ldots < h_{k^A}^A = m, 1 \le k^A \le m$ is *a partition of the string A*.

The sequence h^{A} divides the string A in the following way:

$A = |a_1 a_2 \ldots a_{h_1^A}|a_{h_1^A+1}\ldots a_{h_2^A}|\ldots|a_{h_{k-1}^A+1}\ldots a_{h_{k^A}^A}| = subst_1^A \ldots subst_{k^A}^A$, where $subst_i^A = a_{h_{i-1}^A+1}\ldots a_{h_i^A}, 1 \le i \le k^A$. A pair $[A, h^A]$ is called *the string with the partition*. $\Omega(subst_r^A)$ is the alphabet of the substring $subst_r^A$.

For example, $\Omega = \{a, b, c, d, e\}$, $A = |abc|dababca|bd|daa|, m = 15$, $h^A = 0, 3, 10, 12, 15$; $subst_1^A = abc, subst_2^A = dababca, subst_3^A = bd, subst_4^A = daa$.

A string $C = c_1 c_2 \ldots c_p, 1 \le p \le m$ is *a restricted subsequence* of the string with the partition $[A, \mathrm{h}^{\mathrm{A}}]$, iff

1. there exists a sequence of indices $1 \le i_1 < i_2 < \ldots < i_p \le m$ such that $a_{i_t} = c_t, 1 \le t \le p$, and
2. if $h_{r-1}^A < i_u, i_v \le h_r^A$ then $c_u \ne c_v$, for all r, $1 \le r \le k^A$,

   (this means that each element of an alphabet $\Omega(subst_r^A)$ can be used in C once at most).

The string C is *a common restricted subsequence* of two strings with partition $[A, \mathrm{h}^{\mathrm{A}}]$ and $[B, \mathrm{h}^{\mathrm{B}}]$ if C is the restricted subsequence of $[A, \mathrm{h}^{\mathrm{A}}]$ and C is the restricted subsequence of $[B, \mathrm{h}^{\mathrm{B}}]$ at once. $|C|$ is the length of the restricted common subsequence.

The string C is *a longest common restricted subsequence* of two strings with partition $[A, \mathrm{h}^{\mathrm{A}}]$ and $[B, \mathrm{h}^{\mathrm{B}}]$ if C is a common restricted subsequence of the maximal length.

For example, $\Omega = \{a, b, c\}, A = |aba|abacac|bab|, m = 12, B = |babc|cac|cbcb|, n = 11$. The string $C = bacb$ is the restricted subsequence, $C' = bacab$ is the longest restricted common subsequence but the string $D = babccbb$ is not the restricted common subsequence for $[A, \mathrm{h}^{\mathrm{A}}]$ and $[B, \mathrm{h}^{\mathrm{B}}]$ as it can be seen in Figure 1. The string $C" = babcacbb$ is the longest common subsequence of the strings $A = abaabacacbab$ and $B = babccaccbcb$ if the partition does not matter.

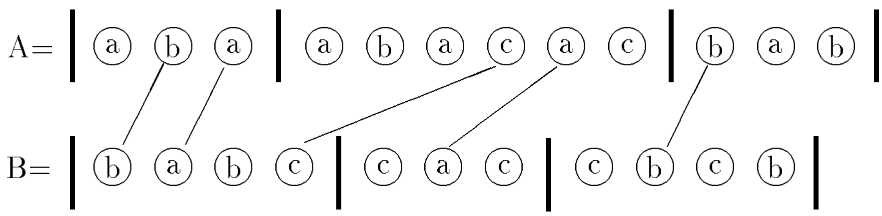


Figure 1. Restricted longest common subsequence of two strings A and B.

A string of sets $\mathcal{A}$ over an alphabet Ω is any finite sequence of sets from $P(\Omega)$. Formally, $\mathcal{A} = A_1 A_2 \ldots A_m, A_i \in P(\Omega), 1 \leq i \leq m$, m is the number of sets in the string $\mathcal{A}$. The length of the symbol string described by $\mathcal{A}$ is $M = \Sigma_{i=1}^{m} |A_i|$.

A string of symbols $C = c_1 c_2 \ldots c_p, c_i \in \Omega, 1 \leq i \leq p$, is *subsequence of symbols* (in short, a subsequence) of string $\mathcal{A}$ if there is nonincreasing mapping $F : \{1, 2, \ldots, p\} \to \{1, 2, \ldots, m\}$, such that

1. if $F(i) = k$ then $c_i \in A_k$, for $i = 1, 2, \ldots, p$
2. if $F(i) = k$ and $F(j) = k$ and $i \neq j$ then $c_i \neq c_j$.

Let $\mathcal{A} = A_1 \ldots A_m, \mathcal{B} = B_1 \ldots B_n, 1 \leq m \leq n$, be two strings of sets over the alphabet Ω. The string of symbols C is *a common subsequence of symbols* of $\mathcal{A}$ and $\mathcal{B}$ is C a subsequence of symbols of $\mathcal{A}$ and C is a subsequence of symbols of the string $\mathcal{B}$. *The longest common subsequence problem* of the strings $\mathcal{A}$ and $\mathcal{B}$ (SSLCS($\mathcal{A}, \mathcal{B}$) consists of finding a common subsequence of symbols C of the maximal length.

The length of SSLCS($\mathcal{A}, \mathcal{B}$) will be denoted $LSSLCS(\mathcal{A}, \mathcal{B})$. Note that C is not unique in general way.

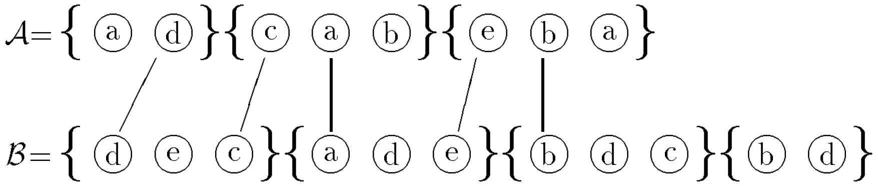


Figure 2. Longest common subsequence of two set strings $\mathcal{A}$ and $\mathcal{B}$.

For example, let $\Omega = \{a,b,c,d,e\}, \mathcal{A} = \{a,d\}\{a,b,c\}\{a,b,e\}, \mathcal{B} = \{c,d,e\}\{a,d,e\}$ $\{b,c,d\}\{b,d\}$. $C = abc$ is a common subsequence of symbols and $C' = adbcb$ and $C'' = dcaeb$ are the longest common subsequences of symbols for $\mathcal{A}, \mathcal{B}$. C'' can be seen in Figure 2.

3 Algorithm for LRCS Problem

Designation.

- $A[i..k] = a_i a_{i+1} \dots a_k$, for $1 \leq i \leq k \leq m$,
- $\langle i, j \rangle$ represents i-th position in the string with the partition $[A, \mathrm{h^A}]$ and j-th position in $[B, \mathrm{h^B}]$, there exist indices r, s such that $1 \leq r \leq k^A, 1 \leq s \leq k^B$ and $h^A_{r-1} < i \leq h^A_r, h^B_{s-1} < j \leq h^B_s$,
- LRCS(A,B) is the longest restricted common subsequence of strings $[A, \mathrm{h^A}]$ and $[B, \mathrm{h^B}]$,
- $LLRCS(A, B)$ is the length of LRCS(A,B),
- $L(i, j) = LLRCS(A[1..i], B[1..j])$.

Principle of the recursive algorithm:

$LLRCS(A, B) = max_{|C|}\{|C| : C$ is the restricted common subsequence of $[A, h^A]$ and $[B, h^B]\}$.

Recursive version of the algorithm is constructed according to the following idea: If an element $c_t = a_{k_t} = b_{l_t}$ is in the $LRCS([A, h^A], [B, h^B])$ then

$$\begin{aligned} LLRCS([A, h^A], [B, h^B]) = 1 + \; & LLRCS([A[1..k_t - 1], h^{A'}], [B[1..l_t - 1], h^{B'}]) + \\ & LLRCS([A[k_t + 1..m], h^{A''}], [B[l_t + 1..n], h^{B''}]), \end{aligned}$$

where $h^{A'}, h^{A''}, h^{B'}, h^{B''}$ are partitions of the related substrings. The recursive version of the algorithm has the exponential time complexity.

A modified Hirschberg's method [6] will be used in the construction of the following time-polynomial algorithm.

A pair $\langle 0, 0 \rangle$ is *a 0-candidate with an empty generating sequence.*

A pair of indices $\langle i, j \rangle, 1 \leq i \leq m, 1 \leq j \leq n, h^A_{r-1} < i \leq h^A_r, h^B_{s-1} < j \leq h^B_s$, will be named *a k-candidate*, $k \geq 1$, iff

1. $a_i = b_j$, and
2. there exists a sequence of pairs which is called *a generating sequence*: $\langle 0, 0 \rangle = \langle i_0, j_0 \rangle, \langle i_1, j_1 \rangle, \dots, \langle i_{k-1}, j_{k-1} \rangle$ such that $i_{k-1} < i$ and $j_{k-1} < j$ and $\langle i_t, j_t \rangle$ is the t-candidate with the generating sequence $\langle i_0, j_0 \rangle, \langle i_1, j_1 \rangle, \dots,$ $\langle i_{t-1}, j_{t-1} \rangle$ and $(a_{i_t} \neq a_i$ or $(i_t \leq h^A_{r-1})$ and $(b_{j_t} \neq b_j$ or $j_t \leq h^B_{s-1})$ for $0 \leq t \leq k-1$.

The set of all k-candidates will be designed $\mathcal{C}_k$ and the generating sequence of k-candidate will be designed I_{k-1}.

For example, $\langle 1,2\rangle, \langle 2,1\rangle, \langle 3,2\rangle, \langle 2,3\rangle, \langle 9,4\rangle, \langle 9,5\rangle, \langle 10,6\rangle \in \mathcal{C}_1$, $\langle 3,2\rangle, \langle 9,4\rangle, \langle 9,5\rangle$, $\langle 10,6\rangle \in \mathcal{C}_2$, $\langle 9,4\rangle, \langle 9,5\rangle, \langle 10,6\rangle \in \mathcal{C}_3, \langle 10,6\rangle \in \mathcal{C}_4, \ldots$ for the strings with partitions $A = |aba|abacac|bab|, B = |babc|cac|cbcb|$.

Remark. $\langle i,j\rangle, h^A_{r-1} < i \leq h^A_r, h^B_{s-1} < j \leq h^B_s$ is 1-candidate with the generating sequence $\langle 0,0\rangle$ if $a_i = b_j$.

Lemma 3.1 *If the pair $\langle i,j\rangle, h^A_{r-1} < i \leq h^A_r, h^B_{s-1} < j \leq h^B_s$ is a k-candidate then $L(i,j) \geq k$.*

Proof. Let k=1 and $\langle i,j\rangle$ is 1-candidate with the generating sequence $\langle 0,0\rangle$. $a_i = b_j$, then $L(i,j) \geq 1$. Let $\langle i,j\rangle$ be a k-candidate. There exist two sequences of indices such that $i_1 < i_2 < \ldots < i_{k-1} < i$ and $j_1 < j_2 < \ldots < j_{k-1} < j$. $\langle i_{k-1}, j_{k-1}\rangle$ is $k-1$-candidate and we suppose $L(i_{k-1}, j_{k-1}) \geq k-1$. $a_i = b_j$ and $a_{i_t} = b_{j_t}$ for $1 \leq t \leq k-1$. The string $C = a_{i_1} a_{i_2} \ldots a_{i_{k-1}} a_i$ is the restricted common subsequence of $A[1..i]$ and $B[1..j]$ because of if $h^A_{r-1} < i_t, i_u \leq h^A_r$ then $a_{i_t} \neq a_{i_u}$ is fulfilled for all $r, 1 \leq r \leq k^A$. Analogously for $B[1..j]$. It follows that $L(i,j) \geq L(i_{k-1}, j_{k-1}) + 1 \geq k$. □

Lemma 3.2 *If $L(i,j) = k$ then there exists k-candidate $\langle i^*, j^*\rangle$ with the generating sequence I_{k-1} such that $i^* \leq i$ and $j^* \leq j$ and $L(i^*, j^*) = k$.*

Proof. If $L(i,j) = k$ then there is the restricted common subsequence $C = c_1 c_2 \ldots c_k$ which is created by elements in the positions determined by sequences $1 \leq i_1 < i_2 < \ldots < i_k \leq i, 1 \leq j_1 < j_2 < \ldots < j_k \leq j$, such that $a_{i_t} = c_t = b_{j_t}$ for $1 \leq t \leq k$, and from the definition of the restricted common subsequence follows:

1. if $h^A_{r-1} < i_u, i_v \leq h^A_r$, then $a_{i_u} \neq a_{i_v}$, for $1 \leq r \leq k^A$, and
2. if $h^B_{s-1} < j_u, j_v \leq h^B_s$, then $b_{j_u} \neq b_{j_v}$, for $1 \leq s \leq k^B$,

Let $i_u, i_v \in \{i_1, \ldots, i_k\}$. The 1. condition can be formulated as *not* $(h^A_{r-1} < i_u, i_v \leq h^A_r)$ or $a_{i_u} \neq a_{i_v}$. The first part means that i_u, i_v are not in the same interval of the partition h^A. If $i_u < i_v \leq h^A_r$ then $i_u \leq h^A_{r-1}$. The condition can be explained $a_{i_v} \neq a_{i_u}$ or $i_u \leq h^A_{r-1}$. Analogously for the condition 2.

Suppose that $i^* = i_k, j^* = j_k$ and $h^A_{r-1} < i_k \leq h^A_r, h^B_{s-1} < j_k \leq h^B_s$. The pair $\langle i_k, j_k\rangle$ is the k-candidate with the generating sequence $\langle 0,0\rangle, \langle i_1, j_1\rangle, \ldots, \langle i_{k-1}, j_{k-1}\rangle$, since $a_{i_k} = b_{j_k}$ and for all $t, 1 \leq t \leq k-1$, the pair $\langle i_t, j_t\rangle$ is the t-candidate with the generating subsequence I_{t-1} and $(a_{i_t} \neq a_{i_k}$ or $i_t \leq h^A_{r-1})$ and $(b_{j_t} \neq b_{j_k}$ or $j_t \leq h^B_{s-1})$. □

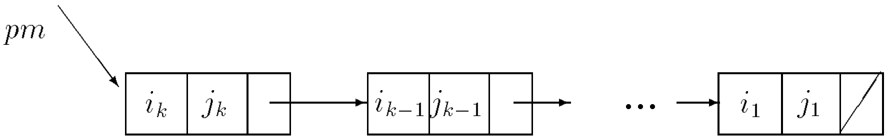
Lemma 3.3 *Let $C = c_1 c_2 \ldots c_k = a_{i_1} a_{i_2} \ldots a_{i_k} = b_{j_1} b_{j_2} \ldots b_{j_k}$ be the longest restricted common subsequence of $A[1..i]$ and $B[1..j]$ and $L(i,j) = k$ is its length. Let $h^A_{r-1} < i+1 \leq h^A_r, h^A_{s-1} < j+1 \leq h^B_s$. Let Cond A is the following condition:*

$a_{i+1} = b_{j+1}$ and $(a_{i+1} \neq a_{i_t}$ or $(i_t \leq h^A_{r-1}))$ and $(b_{j+1} \neq b_{j_t}$ or $(j_t \leq h^B_{s-1}))$ for all $t, 1 \leq t \leq k$.

If the Cond A is fulfilled then $\langle i+1, j+1\rangle$ is (k+1)-candidate and $L(i+1, j+1) = L(i,j)+1$, and the longest restricted common subsequence is $C^* = c_1 c_2 \ldots c_k a_{i+1}$. If Cond A is not fulfilled then L(i+1,j+1)=max $\{L(i, j+1), L(i+1, j)\}$ and the longest restricted common subsequence is in the same form as for max$\{L(i,j+1), L(i+1,j)\}$.

Proof. Suppose that *Cond A* is fulfilled. The sequence $\langle i_1, j_1\rangle, \ldots, \langle i_k, j_k\rangle$ is the generating sequence for (k+1)-candidate $\langle i+1, j+1\rangle$ since $i_k < i+1, j_k < j+1$, and for all $t, 1 \leq t \leq k$ the pair $\langle i_t, j_t\rangle$ is t-candidate with the generating subsequence I_{t-1} and $(a_{i_t} \neq a_{i+1}$ or $i_t \leq h^A_{r-1})$ and $(b_{j+1} \neq b_{j_t}$ or $j_t \leq h^B_{s-1})$. If assumptions of lemma are not fulfilled then $\langle i+1, j+1\rangle$ is not (k+1)-candidate and $L(i+1, j+1)$ can not be greater than $L(i, j+1)$ or $L(i+1, j)$. □

Lemma 3.3 is the base for the construction of the algorithm for a computing of the restricted longest common subsequence of two strings with partitions. We use the dynamic data structure for the construction of linear lists representing the generating sequences of k-candidates, $k = 1, 2, \ldots$ as follows:



Algorithm will work with the following data types

```
{Omega is an alphabet of strings;}
type vertex = record            {element of generating sequence}
                 x, y : integer; {indices}
                 p: pointer;
              end;

type pointerv = ^vertex;        {pointer to the element of
                                  the generating sequence }

type  genseq = record           {record of the length and pointer}
                  length: integer;   {to the generating sequence}
                  pt: pointer;
               end;
```

The definition of the k-candidate gives the method for the construction of the k-candidate if the generating sequence is known. The next function *Candidate* finds if the element $\langle i, j\rangle$ is a potential k-candidate with a generating subsequence with pointer *pm*.

```
function Candidate(pm: pointer; ab: Omega; uA, uB: integer): Boolean;

{It returns the value "true" if <i,j> is a potential k-candidate
 else returns "false".

 pm - pointer to the generating subsequence,
 ab - the candidate in positions i, j,
 uA, uB - upper bounds of intervals for current positions
         i, j: uA<=i, uB<=j.}

var pp:pointerv; q: Boolean; ii, jj:integer;
```

```
begin
   pp:= pm; q:=true;
   while (pp<>nil) and q do
   begin
      ii:=pp^.x; jj:=pp^.y;
      if   (a[ii]=ab) and (ii>=uA)
        or (b[jj]=ab) and (jj>=uB) then q:=false;
      pp:= pp^.p
   end;
   Candidate:= q;
end; {Candidate}
```

Lemma 3.4 *The function* Candidate *computes the value* true *if* $\langle i, j\rangle$ *is a potential k-candidate else the value* false *in* $O(k)$*-time.*

Proof. pm is a pointer to the generating sequence of pairs $\langle i, j\rangle$, $\langle i, j\rangle$ is k-candidate. The function *Candidate* computes the value *false* if in this sequence there exists $\langle i^*, j^*\rangle$ such that $a_{i^*} = a_i = b_j$ and $i^* \geq uA$ or $b_{j^*} = a_i = b_j$ and $j^* \geq uB$. It means that the condition of k-candidate for $\langle i, j\rangle$ is not fulfilled. In the other case *Candidate* gives the value *true*, $\langle i, j\rangle$ is k-candidate with the given generating sequence. Time complexity is $O(k)$ because of each element of the generating sequence is compared with a_i k-times in the worst case. $\square$

The function *Candidate* is used in the algorithm for computing a longest restricted common subsequence of two strings with some partitions.

ALGORITHM A:
{Algorithm constructs a longest restricted common subsequence of two strings with partitions.}

Input: $[A, hA], [B, hB]$ - two strings of symbols with partitions over alphabet *Omega*;
Output: pptr - pointer to the longest restricted common subsequence of A and B;
Variables:

Arrays C, D[0..m] of the type *genseq*.
C[i], D[i] - pointers to the longest common subsequences of A[1..i] and B[1..j];
$hA[1..kA], hB[1..kB]$ - arrays of partitions of the strings A and B;
uA, uB - upper bounds of intervals for current positions $i, j : uA \leq i, uB \leq j$.
dA, dB - the recent numbers of intervals in the partitions,
pp - a pointer to the vertex.

Method:

```
begin
   for i:=0  to n do
   begin
      D[i].pt:=nil; D[i].length:=0;
   end;
   C[0].pt:=nil; C[0].length:=0;
```

```
   dA:=1; uA:=1;
   for i:=1 to m do
      begin
         if i>hA[dA] then begin inc(dA); uA:=hA[dA-1]+1 end;
          dB:=1; uB:=1;
          for j:=1 to n do
          begin
             if j>hB[dB] then begin inc(dB); uB:=hB[dB-1]+1 end;
             if a[i]=b[j] then q:=Candidate(D[j-1].pt,a[i],uA,uB)
                          else q:=false;
             if q then
             begin
                new(pp);
                pp^.p:=D[i-1].pt; pp^.x:=i; pp^.y:=j;
                C[i].pt:=pp; C[i].length:=D[i-1].length+1;
             end  else
                if D[i].length>=C[i-1].length then C[i]:=D[i]
                                              else C[i]:=C[i-1];
           {Invariant1}
           end;
          for j:=1 to n do D[j]:=C[j];
          {Invariant2}
       end;
       len := C[n].length; pptr := C[n].pt;

{ "len" contains the length of the longest restricted common
  subsequence and C[n].pt contains pointer to the LRCS(A,B)}

          writeln('Length LRCS(A,B) =', len:3);
          while pptr<>nil do
          begin
               write(pptr^.x:3,pptr^.y:3,'**');
               pptr:=pptr^.p
          end;
end;
```

Theorem 3.1 *The Algorithm A computes correctly $LRCS(A,B)$ in $O(m \cdot n \cdot p)$-time and $O(n+r)$-space, where p is the length of $LRCS(A,B)$ and $r = |\{\langle i,j\rangle : a_i = b_j, 1 \leq i \leq m, 1 \leq j \leq n\}|$.*

Proof. We specify the invariants of the cycles in the algorithm A.
Invariant1:
$C[j']$ contains the length and the pointer to the $LCSS(A[1..i], B[1..j'])$, for $1 \leq j' \leq j$, and $C[j^*]$ contains the length and the pointer to the $LRCS(A[1..i-1], B[1..j^*])$ for $j < j^* \leq n$.
Invariant2:
$C[j], D[j]$ contains the length and the pointer to the $LRCS(A[1..i], B[1..j])$ for $1 \leq j \leq n$ and $i \leq n$.

The correctness of the algorithm follows immediately from the Invariant1 and Invariant2.

Time complexity: The function *Candidate* requires $O(k)$ steps, $k \leq p$, and it can be repeated at most $m \cdot n$ times. Thus, total time is $O(m \cdot n \cdot p)$.

Space complexity: The arrays C, D require $O(n)$ space, strings $[A, h^A]$ and $[B, h^B]$ require $O(m+n)$ space. If $a_i = b_j$ then function *Candidate* can give a value *true* and in this case a next element is added to the dynamic data structure that requires $O(r)$ space. If $m \leq n$ then the algorithm requires $O(n+r)$ space. □

Let $\mathcal{C}_k$ be the set of all k-candidates, for some $k \geq 1$. *Partial ordering* "$\ll$" can be defined on $\mathcal{C}_k$ in the following way:
$\langle i, j \rangle \ll \langle i^*, j^* \rangle$ iff $i \leq i^*$ and $j \leq j^*$, for $\langle i, j \rangle, \langle i^*, j^* \rangle \in \mathcal{C}_k$.

An element $\langle i, j \rangle$ is a *minimal k-candidate* iff for all $\langle i^*, j^* \rangle \in \mathcal{C}_k, \langle i^*, j^* \rangle \neq \langle i, j \rangle$ is $i^* < i$ or $j^* < j$.

The set of all minimal k-candidates for $k \geq 1$, will be designed $\mathcal{C}_k^{min}$.

Remarks. It is clear that

1. $\mathcal{C}_1 \supseteq \mathcal{C}_2 \supseteq \ldots \supseteq \mathcal{C}_p \supseteq \mathcal{C}_{p+1} = \emptyset$
2. $\mathcal{C}_1^{min} \neq \mathcal{C}_2^{min} \neq \ldots \neq \mathcal{C}_p^{min}$
3. Let $1 \leq k \leq p, \langle i, j \rangle \in \mathcal{C}_k$ and $\langle i, j \rangle \notin \mathcal{C}_{k+1}$ then $L(i, j) = k$.

Hirschberg's method of minimal k-candidates [6] can be applied in this special case of strings with partitions and gives $O(n \cdot p^2)$-time algorithm, where p is the length of the longest restricted common subsequence.

4 Transformation of SSLCS Problem to LRSC Problem

Let $\mathcal{A} = A_1 A \ldots A_m, 1 \leq m$ be the string of the sets over Ω. Elements of a subset $A_i, A_i \in P(\Omega), 1 \leq i \leq m$, can be chosen in an arbitrary order and there are $|A_i|!$ permutations of these elements.

Let $p(A_i)$ be a permutation of elements inA_i (it is a string consisting of all symbols in A_i).

We define a string of symbols A in the following way:

$$A = p(A_1)p(A_2) \ldots p(A_m), \tag{1}$$

A is the concatenation of strings $p(A_1), p(A_2) \ldots, p(A_m)$.

Let $\mathbf{A}$ be the set of all strings of symbols created by (1). The number of elements in $\mathbf{A}$ is $|\mathbf{A}| = \Pi_{i=1}^{|A|} |A_i|!$. Let the elements in $\mathbf{A}$ are enumerated in some way, $\mathbf{A} = \{A^i\}, i = 1, \ldots, |\mathbf{A}|$.

Analogously, it is possible to construct the set $\mathbf{B}$ to the string $\mathcal{B}$. Let be

$$L(\mathbf{A}, \mathbf{B}) = max \{LLCS(A^i, B^j) : 1 \leq i \leq |\mathbf{A}|, 1 \leq j \leq |\mathbf{B}|\}. \tag{2}$$

Lemma 4.1 $L(\mathbf{A}, \mathbf{B}) = LSSLCS(\mathcal{A}, \mathcal{B})$.

Proof. Let $1 \leq i \leq |\mathbf{A}|, 1 \leq j \leq |\mathbf{B}|$. $LLCS(A^i, B^j)$ is the length of the longest common subsequence of strings of symbols A^i and B^j. Both strings are constructed as a special cases of strings $\mathcal{A}, \mathcal{B}$, and $LLCS(A^i, B^j) \leq LSSLCS(\mathcal{A}, \mathcal{B})$, for $1 \leq i \leq |\mathbf{A}|, 1 \leq j \leq |\mathbf{B}|$. It means $L(\mathbf{A}, \mathbf{B}) = max_{i,j}\{LLCS(A^i, B^j\} \leq LSSLCS(\mathcal{A}, \mathcal{B})$. Since all possible strings A^i and B^j have been used, the following inequality holds $L(\mathbf{A}, \mathbf{B}) \geq LSSLCS(\mathcal{A}, \mathcal{B})$. □

Let $1 \leq k \leq m$. $p^*(A_k)$ is constructed from $p(A_k)$ by adding some elements of A_k into arbitrary positions of $p(A_k)$. Each element of A_k is in the $p^*(A_k)$ once at least.

Lemma 4.2 *Let i, j be indices such that $L(\mathbf{A}, \mathbf{B}) = LLCS(A^i, B^j)$, $A^i = p(A_1)p(A_2) \dots p(A_m)$. Let $1 \leq k \leq m$ and $A^{i*} = p(A_1) \dots p(A_{k-1})p^*(A_k)p(A_{k+1}) \dots p(A_m)$. If each element of A_k can be chosen from $p^*(A_k)$ once at most then $L(\mathbf{A}, \mathbf{B}) = LLCS(A^{i*}, B^j)$.*

Proof. Since each element of A_k can be chosen from $p^*(A_k)$ once at most (some permutation of elements in A_k), we have $L(\mathbf{A}, \mathbf{B}) \geq LLCS(A^{i*}, B^j)$. $p^*(A_k)$ has been constructed by adding some elements to $p(A_k)$ and the following inequality is fulfilled: $LLCS(A^{i*}, B^j) \geq LLCS(A^i, B^j)$. □

Lemma 4.3 *Let i, j be indices such that $L(\mathbf{A}, \mathbf{B}) = LLCS(A^i, B^j)$. Let $A^{i*} = p^*(A_1) \dots p^*(A_m), B^{j*} = p^*(B_1) \dots p^*(B_n)$. If each element of $A_k, 1 \leq k \leq m$ can be chosen from $p^*(A_k)$ once at most and each element of $B_t, 1 \leq t \leq n$ can be chosen from $p^*(B_t)$ once at most then $L(\mathbf{A}, \mathbf{B}) = LLCS(A^{i*}, B^{j*})$.*

Proof. A^{i*} and B^{j*} are constructed by adding some elements to the strings A^i, B^j and thus $LLCS(A^{i*}, B^{j*}) \geq LLCS(A^i, B^j)$. Since each part $p^*(A_k)$, or $p^*(B_t)$ can be used as a permutation of A_k or B_t respectively, we have $L(\mathbf{A}, \mathbf{B}) \geq LLCS(A^i, B^j)$. Thus $L(\mathbf{A}, \mathbf{B}) = LLCS(A^i, B^j)$. □

Let $\mathcal{A} = A_1A_2 \dots A_m, m \geq 1$ be the string of the sets over Ω. Let $p^+(A_k), 1 \leq k \leq m$ be the string of all permutations of A_k (permutations of elements in A_k are in $p^+(A_k)$ as the subsequences). Let $A^* = p^+(A_1)p^+(A_2) \dots p^+(A_m)$. Analogously for $\mathcal{B}$, $B^* = p^+(B_1)p^+(B_2) \dots p^+(B_n)$.

Theorem 4.1 *$L(\mathbf{A}, \mathbf{B}) = LLCS(A^*, B^*)$ if each element of A_k , respectively B_t, can be used once at most from the part $p^+(A_k)$, respectively $p^+(B_t)$.*

Proof. There are the indices i, j such that $L(\mathbf{A}, \mathbf{B}) = LLCS(A^i, B^j)$. According to Lemma 4.3 $LLCS(A^i, B^j) = LLCS(A^{i*}, B^{*j})$. The strings A^*, B^* are some special cases of strings A^{i*}, B^{j*} and it implies $L(\mathbf{A}, \mathbf{B}) = LLCS(A^*, B^*)$. □

Lemma 4.4 *The length of the string A^* is less or equal than M^2, the length of B^* is less or equal than N^2.*

Proof. $p+(A_k)$ can be constructed by a repeating of A_k $|A_k|$ times. This construction gives the length $|A_k|^2$. In [12] is presented the construction of shorter string with the length $|A_k|^2 - 2 \cdot |A_k| + 4, |A_k| \geq 4$. The length of A^* is $|A^*| = \Sigma_{k=1}^m |p^+(A_k)| \leq \Sigma_{k=1}^m |A_k|^2 \leq (\Sigma_{k=1}^m |A_k|)^2 = M^2$. Analogously, $|B^*| \leq N^2$. □

For example, let $\Omega = \{a, b, c, d, e\}, \mathcal{A} = \{a, d\}\{a, b, c\}\{a, b, e\}, \mathcal{B} = \{c, d, e\}\{a, d, e\}$ $\{b, c, d\}\{b, d\}$. It is possible to construct the following strings with partitions $[A^*, h^{A^*}]$ and $[B^*, h^{B^*}]$ to $\mathcal{A}$ and $\mathcal{B}$ respectively:

$A^* = |ada|cabcacb|ebaebea|, h^{A^*} = 0, 3, 10, 17, k^{A^*} = 3,$

$B^* = |decdedc|adeadae|bdcbdbc|bdb|, h^{B^*} = 0, 7, 14, 21, 24, k^{B^*} = 4.$

And the longest common subsequence of $\mathcal{A}$ and $\mathcal{B}$ can be computed by the algorithm for the restricted common subsequence problem of the strings with partitions $[A^*, h^{A^*}]$ and $[B^*, h^{B^*}]$: $LSSLCS(\mathcal{A}, \mathcal{B}) = LLRCS([A^*, h^{A^*}], [B^*, h^{B^*}])$.

Theorem 4.2 *Set-Set LCS Problem for two strings of sets can be computed in $O(M^2 \cdot N^2 \cdot p)$ time and $O(N^2 + r)$ space, where M, N are the numbers of symbols in subsets $\mathcal{A}$ or $\mathcal{B}$, respectively, p is the length of the longest common subsequence and $r = |\{\langle i, j\rangle : a_i = b_j, a_i \in A^*, b_j \in B^*, 1 \leq i \leq M^2, 1 \leq j \leq N^2\}|$.*

Proof. It follows from the Lemmas 4.2, 4.3, 4.4 and Theorem 4.1. □

5 Concluding Remarks

The polynomial algorithm for the solution of the LRCS Problem with a restricted using of elements has been presented. The algorithm can be used to show in the very simple way that SSLCS Problem has a polynomial complexity.

The LRCS Problem offers a generalization that is leading to the following problem: Let $[A, h^A], [B, h^B]$ be two strings with the partitions and with the restricted using of elements, let f_A, f_B are integer functions called weights of elements in positions: $f_A, f_B : \Omega \times \{1, 2, \dots, n\} \to Integer$. For example, $A = abacbda$ the function f_A can have values $f_A(a, 3) = 7, f_A(a, 7) = 4, \dots$. The measure of a common subsequence is the sum of weights of the matching elements. A weight of matching elements is the sum (or maximum) of weights of these elements in strings A and B in matching positions. Construct restricted common subsequence with the maximal measure.

References

[1] Andrejková, G.: *Systolic systems for the longest common subsequence problem.* Computers and Artificial Intelligence, 5 (1986), No. 3, p. 199–212.

[2] Apostolico, A.: *Improving the worst-case performance of the Hunt-Szymanski strategy for the longest common subsequence of two strings.* Information Processing Letters 23 (1986), p. 63–69.

[3] Dewar, R. B., Merritt, S. M., Sharir, M.: *Some modified algorithms for Dijkstra's longest common subsequence problem.* Acta Informatica 18, 1982, p. 1–15.

[4] Heckel, P.: *A technique for isolating differences between files.* Comm. ACM 21, 4 (Apr. 1978), p. 264–268.

[5] Hirschberg, D. S.: *A linear space algorithms for computing maximal common subsequences.* Comm. ACM 18, 6 (June 1975), p. 341–343.

[6] Hirschberg, D. S.: *Algorithms for longest common subsequence problem.* Journal ACM 24, 4 (Oct 1977), p. 664–675.

[7] Hirschberg, D. S., Larmore, L. L.: *The Set LCS Problem.* Algorithmica 2 (1987), p. 91–95.

[8] Hirschberg, D. S., Larmore, L. L.: *Set-Set LCS Problem.* Algorithmica 4 (1989), p. 503–510.

[9] Huang, S. S., Asuri, S. H.: *Algorithms for the Set-LCS and Set-Set-LCS Problems.* Tech. Report No. UH-CS-89-09, University of Houston, March, 1989.

[10] Hunt, J. W., Szymanski, T. G.: *A fast algorithm for computing longest common subsequences.* Comm. ACM 20, 5 (May 1977), p. 350–351.

[11] Lowrance, R., Wagner, R. A.: *An extension of the string-to-string correction problems.* Journal ACM 22, 2 (Apr. 1975), p. 177–183.

[12] Mohanty, S. P.: *Shortest string containing all permutations.* Discrete Mathematics 31, 1980, p. 91–95.

[13] Nakatsu, N., Kombayashi, Y., Yajima, S.: *A longest common subsequence algorithm suitable for similar text strings.* Acta Informatica 18, 1982, p. 171–179.

[14] Needleman, S. B., Wunsch, Ch. D.: *A general method applicable to the search for similarities in the amino acid sequence of two proteins.* Journal Mol. Biol. 48, 1970, p. 443–453.

[15] Robert, Y., Tchuante, M.: *A Systolic array for the longest common subsequence problem.* Information Processing Letters 21 (1985), p. 191–198.

[16] Robert, Y., Tchuante, M.: *Calcul en temps linéaire d'une plus longue sous-suite commune à deux chaîne sur une architecture systolique.* C. R. Acad. Sci. Paris, Série I, No. 7, 1984, p. 269–271.