Minimum Height and Sequence Constrained Longest Increasing Subsequence

Chiou-Ting Tseng, Chang-Biau Yang, Hsing-Yen Ann
Department of Computer Science and Engineering
National Sun Yat-sen University
Taiwan, R.O.C.
cbyang@cse.nsysu.edu.tw

Abstract

Given a string $S = a_1a_2a_3\cdots a_n$, the longest increasing subsequence (LIS) problem is to find a subsequence of S such that the subsequence is increasing and its length is maximum. In this paper, we propose and solve two variants of the LIS problem. The first one is the minimum height LIS where the height means the difference between the largest and smallest elements. We propose an algorithm with $O(n \log n)$ time and $O(n)$ space for solving it. The second one is the sequence constrained LIS (SCLIS) that given a string S and a constraint C, we are to find the LIS of S containing C as its subsequence. We propose an algorithm with $O(n \log(n+|C|))$ time for solving it. And then we solve the SCLIS with preprocessing. We propose a preprocessing algorithm with $O(n^2 \log \log n)$ time on S so that with a given sequence of positions, we can answer the SCLIS query in $O(|C|+|OUTPUT|)$ time where the constraint is the subsequence on the given positions of S.

Keywords: Algorithm, Longest Increasing Subsequence, Height, Constraint.

1 Introduction

Given a string $S = a_1a_2a_3\cdots a_n$, an *increasing subsequence* (IS) is a subsequence [23] $a_{i_1}a_{i_2}a_{i_3}\cdots a_{i_k}$ that $a_{i_p} < a_{i_q}$ if $i_p < i_q$, for $1 \le p < q \le k$. For example, consider $S = 41573$, three of its increasing subsequences are 13, 457 and 157. The *longest increasing subsequence* (LIS) problem is to find the longest among all increasing subsequences. Note that the LIS of a given string may not be unique. For example, both 457 and 157 are LIS's of $S = 41573$.

The LIS problem is interesting in both combinatorial perspective, such as pattern recognition, and biological applications. Delcher et al. [8] used LIS to help finding the whole genome alignment. A straightforward method of finding the LIS is to obtain the longest common subsequence of the input string and the sorted input string, with time complexity $O(n^2)$. Schensted [19] is the first one who defined the LIS problem and proposed an algorithm with $O(n \log n)$ time. Hunt and Szymanski [10] improved the algorithm to $O(n \log \log n)$ time. And later, many papers [1][3-5][15][16][26] studied the LIS problem by using the van Emde Boas priority queue [22], which supports insertion, deletion, finding, predecessor, and successor operations in $O(\log \log |\Sigma|)$ time, where Σ is the alphabet set of the input string. In LIS, if the input is an integer string where each integer is in $\{1, 2, 3,\cdots, n\}$, then $|\Sigma| = n$, and the LIS algorithm needs only $O(n \log \log n)$ time since $|\Sigma| = n$. The longest decreasing subsequence problem can also be solved similarly. The length distribution of the LIS has been analyzed by Aldous and Diaconis [2]. In their result, the average length of the LIS of a string with length n is about $2\sqrt{n}$.

Kim showed that finding the LIS is equivalent to finding the maximum independent set in a permutation graph [12]. A permutation graph has no duplicated symbol, but the input string of the LIS problem might have. It seems that the duplicated symbols in the input string do not affect the complexity of the algorithm. After a preprocessing with $O(n)$ time, the duplicated symbols can be mapped to other symbols which would not affect the resulting of increasing subsequence or the decreasing subsequence. In the preprocessing, when we come to a symbol a with its ith occurrence, we change it to a-$i\varepsilon$ in the increasing subsequence case and to a+$i\varepsilon$ in the decreasing subsequence case for small ε.

Various variants of the LIS problem have also been discussed [4][5][11][18][21][24][25]. In this paper, we first define two LIS variants and then propose effcient algorithms to solve them. The first one is the minimum height LIS where the height means the difference between the largest and smallest elements found in the solution. The second one is the sequence constrained LIS (SCLIS) that given a string S and a constraint C, we are to find the LIS of S containing C as its subsequence.

The rest of this paper is organized as follows. In Section 2, we will review some other previous work related to the LIS problem. In Section 3, we solve the minimum height LIS problem. In Section 4, we solve the sequence constrained LIS problem. Finally, Section 5 gives conclusions and some future work.

2 Previous Results

Finding the LIS in streaming data has the limitation that the passed data can only be retained a limited number of times, Liben-Nowell et al. [14] gave an algorithm with $\log(1+\frac{1}{\varepsilon})$ passes, $O(\log l)$ or $O(\log \log |\Sigma|)$ updating time, and $O(l^{1+\varepsilon} \log |\Sigma|)$ space, where l is the length of the LIS.

A variant of the LIS problem is to find the *heaviest increasing subsequence* (HIS). Given a string S formed by $\sum$, where each symbol α in $\sum$ has a weight $w(\alpha)$, the weight of a subsequence is the sum of the weights of all symbols contained in the subsequence. The HIS problem is to find the increasing subsequence with the maximum weight. As the equivalence of LIS and the maximum independent set of the permutation graph, HIS is equivalent to the maximum weight independent set of the permutation graph. Several papers [6][9][13][26] have devoted to the study of the maximum weight independent set problem in graphs, including permutation graphs.

A simple extension of the LIS problem is to find the LIS of every substring. In our previous work [21], we design an effcient preprocessing method, with $O(n^2)$ time, to solve it. After the preprocessing has been performed, the required answering time is linear to the output size.

Another extension of the LIS problem is the *longest common increasing subsequence* (LCIS) problem. Given two strings $A = a_1a_2a_3 \cdots a_m$, $B = b_1b_2b_3 \cdots b_n$ where each pair of symbols in A and B are comparable, the common increasing subsequence of A and B is $G = g_1g_2g_3 \cdots g_l$ where $g_1 = a_{i_1} = b_{j_1}$, $g_2 = a_{i_2} = b_{j_2}$, $\cdots$, $g_l = a_{i_l} = b_{j_l}$ and for all $1 \le p < q \le l$, $i_p < i_q$, $j_p < j_q$, $g_p < g_q$. The LCIS of A and B is the longest among all common increasing subsequences of A and B. Yang et al. [25] proposed an algorithm for solving this problem in $O(n^2)$ time. In 2005, several papers tightened the upper bound. Katriel and Kutz [11] gave an algorithm with $O(nl \log n + Sort)$ time, where *Sort* is the time required for sorting string B into nondecreasing order. Chan et al. [5] gave an algorithm with $O(\min(r \log l, nl+r) \log \log n+Sort))$ time, where r is the number of matched pairs between A and B. Brodal et al. [4] gave an algorithm with $O((m + nl) \log \log |\sum| + Sort)$ time. For small $\sum$, the algorithm has a tighter bound $O(m)$ when $|\sum| = 2$, $O(m + n \log n)$ when $|\sum| = 3$. Yoshifumi [18] gives a linear space algorithm for the LCIS problem.

For the LCIS of multiple sequences, Chan et al. [5] gave an algorithm with $O(min(Nr^2, Nr \log p \log Nr) + NSort_{\sum}(n))$ time, where N is the number of input sequences, and $Sort_{\sum}(n)$ denotes the time required for sorting all sequences. Brodal et al.[4] proposed an algorithm with $O(min(Nr^2, r \log^{N-1} r \log \log r)+NSort_{\sum}(n))$ time.

Yang et al. [24] proposed the constrained LIS problem. They defined two types of constraints, the first one is that the difference between two neighboring elements in the increasing subsequence must be in $[L_V , U_V]$ and their positional distance in the original string must be in $[L_I , U_I]$. We call the difference between two neighboring elements as the *cliff* in this paper. They proposed an algorithm with $O(n \log(U_I - L_I))$ time and $O(n)$ space. The second constraint stipulates that the slope of two neighboring elements in LIS must be greater than a predefined value, where the slope is defined as their difference divided by their positional distance in the original string. They solved it in $O(n \log r)$ time and $O(n)$ space where r is the output size.

3 Minimum Height LIS

The *height* of an increasing sequence is defined as the difference between the largest and the smallest elements. In fact, the minimum height constraint is the minimum sum of the cliff constraint. Given a string $S = a_1a_2 \cdots a_n$, the *minimum height LIS* (MHLIS) problem is to find an LIS with the minimum height. For example, suppose S = 4683571, then its MHLIS is 457 or 467.

The *representative increasing subsequence* (RIS) of a string S is the principle row of the row tower, defined by Albert et al. [1]. The ith element in RIS of S is the minimum ending number of increasing subsequences with length i. For example, S = 41573 has increasing subsequences {4, 1, 5, 7, 3}, {45, 47, 15, 17, 13, 57} and {457, 157}. The ending numbers of increasing subsequences with length 2 are {5, 7, 3}, thus the minimum is 3. So the RIS of S is 137. Note that the RIS may not be a subsequence of the original string. If an element is smaller than the ith element of the RIS, it can not be the ending number of increasing subsequence with length i+1. So the length of the LIS ending at a certain element x can be found by searching for the largest element that is smaller than x in the previous RIS, and then adding one to the position. Likewise, insertion of a new element x to RIS R is to find the minimum element in R greater than or equal to x and replace it by x. If x is greater than all elements in R, then we append x to the end of R, so its length is increased by one.

Let l_i denote the length of the LIS ending at a_i. We call an IS ending at a_i maximal if its length is l_i. We record the maximal ending number smaller than a_i of maximal IS's with length $l_i - 1$ in $a_1a_2 \cdots a_{i-1}$ as the previous element of a_i in MHLIS. We will prove the correctness of this approach in the following.

Our algorithm for finding the MHLIS is given as follows.

Step 1: Maintain l balanced binary search trees $q_1, q_2, \cdots, q_l$, where each q_j , $1 \le j \le l$, records the ending numbers of all maximal IS's with length j, and l denotes the length of the LIS.

Step 2: Whenever an element a_i is read, find the LIS ending at a_i by the following steps.

Step 2.1: If $i = 1$, we add a_1 into q_1 and set RIS to a_1. Otherwise, perform binary search on the previous RIS to find the smallest element greater than or equal to a_i. The position index of the found element is the length l_i of the LIS ending at a_i.

Step 2.2: Add a_i to q_{l_i} . If l_i is greater than 1, suppose the predecessor of a_i in q_{l_i-1} is p_i, record the predecessor of a_i in the MHLIS ending at a_i as p_i.

Step 2.3: Insert a_i into the previous RIS.

For example, for S = 4683571, the result is shown in Figure 1. When we are going to add 7, the previous RIS is 358. 7 will replace 8 which is in position 3, and the current RIS becomes 357. We add 7 to q_3 and the largest element smaller than 7 in q_2 is 6, so we set the predecessor of 7 to 6. The minimum height of the increasing subsequences ending at 7 is the height of 6 which is 2 plus the additional height of 7 − 6 = 1, so the total height is 3.

Now, we prove the correctness of our algorithm.

Theorem 1. *Our approach finds the MHLIS ending at* a_i.

Proof. Suppose the LIS we find is $b_1b_2 \cdots b_{l_i-1}a_i$. If we say that $d_1d_2 \cdots d_{l_i-1}a_i$ has smaller height, we have $d_k \geq b_{k+1}$, d_k is not on the right of b_k. Here we use the term "not on the right" because b_k may be the same as d_k. We will show this by induction.

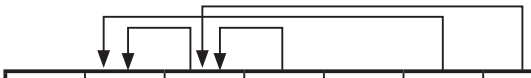
For k = 1, we have $a_i - d_1 < a_i - b_1$, so $d_1 > b_1$. If d_1 is on the right of b_1, then $b_1d_1d_2 \cdots d_{l_i-1}a_i$ is an IS with length l_i+1, which is a contradiction. Thus, d_1 is not on the right of b_1. Also, b_1 is on the left of b_2, so d_1 is on the left of b_2. Suppose $d_1 < b_2$, and we have $d_1 > b_1$, then b_2 would choose d_1 instead of b_1, which is a contradiction. So $d_1 \geq b_2$.

Suppose the assumption holds for $k = m$, that is $d_m \geq b_{m+1}$ and d_m is not on the right of b_m. For $k = m + 1$, $d_{m+1} > b_{m+1}$ because $d_m \geq b_{m+1}$, $d_{m+1} > d_m$. If d_{m+1} is on the right of b_{m+1}, then $b_1b_2 \cdots b_{m+1}d_{m+1} \cdots d_{l_i-1}a_i$ is an IS with length l+1, which is a contradiction. So d_{m+1} is not on the right of b_{m+1}. Also, b_{m+1} is on the left of b_{m+2}, so d_{m+1} is on the left of b_{m+2}. Suppose $d_{m+1} < b_{m+2}$, and we have $d_{m+1} > b_{m+1}$, then b_{m+2} would choose d_{m+1} instead of b_{m+1}, which is a contradiction. So $d_{m+1} \geq b_{m+2}$. By the induction hypothesis, this assumption is true for all k.

Recall that we assume there exists $d_1d_2 \cdots d_{l_i-1}a_i$ with smaller height. From above, we have $d_{l_i-2} \geq b_{l_i-1}$, and $d_{l_i-1} > d_{l_i-2}$, so $d_{l_i-1} > b_{l_i-1}$. But since we select b_{l_i-1} instead of d_{l_i-1}, we have $b_{l_i-1} \geq d_{l_i-1}$ which is a contradiction. So there is no other IS of length l_i with smaller height □.

By Theorem 1, we can find the MHLIS ending at a certain element. We maintain the minimum height element in every binary search tree. To get the MHLIS, we start from the minimum height element in q_l and trace back the LIS by continuously switching to the predecessor of the current element.

The time complexity is analyzed as follows. For each a_i, we spend O(log n) time on deciding the length l of the LIS by binary search on the RIS, and O(1) time on inserting a_i into the RIS. Finding predecessor in the set of ending numbers of IS's with length l − 1 takes O(log n) time. Inserting into the set of ending numbers of length l IS's and the RIS both takes O(log n) time. Finally, calculating the height of the LIS ending at a_i takes constant time if the predecessor and the height of the LIS ending at the predecessor are given. Tracing out the predecessor takes O(n) time by following the predecessor link. So totally we need O(n log n) time to find the MHLIS. And the space requirement is O(n).



	4	6	8	3	5	7	1
R/S	4	46	468	368	358	357	157
L=1	4	4	4	34	34	34	134
L=2		6	6	6	56	56	56
L=3			8	8	8	78	78
H	0	2	4	0	1	3	0

Figure 1 The Minimum Height LIS for S = 4683571

4 Sequence Constrained LIS

Given a string $S = a_1a_2 \cdots a_n$ and an increasing constraint $C = c_1c_2 \cdots c_k$, the *sequence constrained longest increasing subsequence* (SCLIS) problem is to find an LIS containing C as its subsequence. For example, if S = 1529367 and C = 59, then the SCLIS is 159 while the LIS is 12367. Because we are finding an IS and the constraint is a subsequence of the IS, the constraint has to also be increasing.

Note that if there is no duplicated symbol in the input string, the problem becomes finding the occurrence of the constraints in the input sequence as follows. Let the constraint C be on positions $p_1, p_2, \cdots, p_k$ in S. To simplify the discussion, we add two dummy constraints $c_0 = -\infty$ in front of S and C and $c_{k+1} = \infty$ at the rear of S and C. Cut the input sequence by p_i for $1 \leq i \leq k$, and find the LIS of each substring starting at p_i and ending at the previous element of p_{i+1} with value also starting at c_i and smaller than c_{i+1}, $0 \leq i \leq k$. Then the IS's are concatenated together. The concatenated IS is the answer. So the lower bound of the time complexity is the time required for finding the LIS.

This problem can be solved by the similar layered approach for solving the *constrained longest common subsequence* (CLCS) problem [7][17][20]. First, we put the symbol c_i as the symbol of the ith floor, where the 0th floor means no constraint is satisfied. Because all constraints need be satisfied, in the region after constraint c_i has been satisfied but constraint c_{i+1} is not satisfied yet, this part of IS in the final SCLIS should be with value larger than or equal to c_i but smaller than c_{i+1}. Otherwise, c_{i+1} is unable to concatenate to the current increasing subsequence. Besides, we only insert an element when all constraints smaller than it are already inserted.

In the *i*th floor, we maintain an RIS R_i of the elements greater than or equal to c_i and less than c_{i+1}. Originally when we insert an element into the RIS, we replace the smallest element larger than or equal to it. But now, if the element we are going to replace is one of the constraints, which is the first element in each layer, we do not replace it.

Our SCLIS algorithm is to insert the elements of S one by one from left to right into our data structure and the algorithm for inserting a new element is given as follows:

Input: The element to be inserted, e, and the constraint C.

Step1: Find the position of e in C. If $e = c_i$, go to step 2.1. If e is between c_i and c_{i+1}, go to step 2.2.

Step 2.1: If R_{i-1} is not NULL, set predecessor of e to the last element of R_{i-1}. If R_{i-1} is not NULL or $i = 1$, insert e into R_i. And skip step 2.2.

Step 2.2: If c_i is already in R_i or $i = 0$, insert e to R_i, set predecessor of e to the predecessor of e in R_i.

Step 2.1 means that we only accept constraint c_i when all previous constraints have been satisfied. For example, in Figure 2, the first 7 can not be added since 3 has not been added yet. In Step 2.2, we only add e to R_i when c_i has appeared. It is based on the same reason, if c_i has not appeared yet, the elements with value greater than c_i can never be in the final SCLIS. Take the first 4 in Figure 2 as an example, it can not be added because 3 is not in R_1 yet.

The time complexity of this approach is analyzed as follows. If we use arrays to implement this approach. Step 1 takes $O(\log |C|)$ time by binary search on C. Step 2.1 takes constant time. Step 2.2 takes $O(\log n)$ time for doing binary search on R_i. So the time required for inserting one element is $O(\log n)$. There are n elements to be inserted, so the total construction time is $O(n \log(n + |C|))$. The output operation can be achieved in $O(n)$ time by continuously lookup the predecessor table. The space complexity of the R's is $O(n)$ because each element is in at most one of the R's and the predecessor table takes $O(n)$ space. Note that if the constraint is NULL, the problem becomes the original LIS problem (without constraint), so the lower bound of the time complexity is the time required for finding the LIS.

Next, we extend this problem to be the preprocess-query variation. Given a string S, we first preprocess it so that we can answer the SCLIS more effciently when a query on the constraint C with positions $p_1, p_2, \cdots, p_k$, is asked.

Our idea to this variation is to concatenate the LIS's between two neighboring constraints with value between the two constraints. Our previous method can be transformed into a preprocessing method easily. We suppose every element is the only constraint and record the RIS's and the predecessors when we add the elements sequentially into every layer. For example, for S_2 = 21624539, the result is shown in Table 1 where X means a_j has no predecessor. Note that in the same column, the predecessors are different, so we need to record them separately.

In the query phase, we first scan the constraint to see if is increasing. If not, we just output the result as an empty string. Otherwise, we start to find out the SCLIS by following the predecessor links starting from the p_kth layer. When we are on the p_jth layer and we reach the first element, which is the constraint of the layer, we jump vertically to the p_{j-1}th layer and continue by following the predecessor link. When we have satisfied all constraints, we use the predecessor link in the RIS layer to continue the trace back. For example, for S_2 in Table 1, if the constraint is the second and fourth element, which is 12. We start from the fourth layer and get 2459, then we jump to the second layer and get 1, then we jump to the RIS layer and get X. So the SCLIS is 12459.

	7	4	3	7	3	1	5	7	9	8
						1				
3			3		3		35			
7				7				7	79	78

Figure 2 The Constrained LIS for S = 7437315798 and C = 37

Table 1 The Constrained RIS Table

	2	1	6	2	4	5	3	9
RIS	2, X	1, X	16, 1	12, 1	124, 2	1245, 4	1235, 2	12359, 5
2	2, X	2, X	26, 2	26, X	24, 2	245, 4	235, 2	2359, 5
1		1, X	16, 1	12, 1	124, 2	1245, 4	1235, 2	12359, 5
6			6, X	6, X	6, X	6, X	6, X	69, 6
2				2, X	24, 2	245, 4	235, 2	2359, 5
4					4, X	45, 4	45, X	459, 5
5						4, X	5, X	59, 5
3							3, X	39, 3
9								9, X

The time complexity of our algorithm is analyzed as follows. Our preprocessing algorithm fills in the table by column major. If we spend $O(n \log n)$ time to sort the elements in the input string. We can map the elements to $\sum = \{1, 2, \cdots, n\}$. Then we can use one van Emde Boas priority queue(vEBpq) to record the RIS's for each row. When we add a new element, we also record the predecessor of the newly added element. We need one predecessor query, one insertion and at most one deletion in each cell, so totally we need $O(n^2 \log \log n)$ time to do the preprocessing. In the query phase, checking if the constraint is increasing takes $O(|C|)$ time. Following the predecessor links to trace back the SCLIS takes $O(|OUTPUT|)$ time. So the total query time is $O(|C|+|OUTPUT|)$, which is optimal. The space complexity is $O(n^2)$ because each vEBpq takes $O(n)$ space and the predecessor table takes $O(n^2)$ space.

5 Conclusion and Future Works

In this paper, we propose and solve two variants of the LIS problem, the minimum height LIS (MHLIS) problem and the sequence constrained LIS (SCLIS) problem. We first propose an algorithm with $O(n \log n)$ time and $O(n)$ space for solving the MHLIS problem. Since the LIS of a given string may not be unique, we propose a possible way to decide which one is better among them. This can be applied to the situation that there are many available boxes and we desire to pack as many as possible by putting one inside another so that the thickness of the boxes is minimized and thus the minimum space is occupied.

For solving the SCLIS problem, we proposed an $O(n \log(n + |C|))$ time algorithm. This can also be applied to the situation similar to the above. That is, we want to pack as many boxes as possible while some specific boxes must be packed in. In addition, we also discuss the online SCLIS problem. We propose a preprocessing algorithm with $O(n^2 \log \log n)$ time so that each query can be answered in $O(|C| + |OUTPUT|)$ time. The possible future work may involve trying to improve the algorithmic effciency or to prove that our algorithm is optimal, solving other cliff based constraints like min-max, max-min, etc. And among all, exploring more applications of the LIS problem, either with or without variant.

References

[1] M. H. Albert, A. Golynski, A. M. Hamel, A. Lopez-Ortiz, S. S. Rao and M. A. Safari, *Longest Increasing Subsequences in Sliding Windows*, *Theoretical Computer Science*, 2004, pp.405-414.

[2] D. Aldous and P. Diaconis, *Longest Increasing Subsequences: From Patience Sorting to the Baik-Deift-Johansson Theorem*, *BAMS: Bulletin of the American Mathematical Society*, Vol.36, 1999, pp.413-432.

[3] S. Bespamyatnikh and M. Segal, *Enumerating Longest Increasing Subsequences and Patience Sorting*, *Information Processing Letters*, Vol.76, No.1-2, 2000, pp.7-13.

[4] G. S. Brodal, K. Kaligosi, I. Katriel and M. Kutz, *Faster Algorithms for Computing Longest Common Increasing Subsequences, Tech*, Rep. BRICS-RS-05-37, BRICS, Department of Computer Science, University of Aarhus, December, 2005.

[5] W. T. Chan, Y. Zhang, S. P. Y. Fung, D. Ye and H. Zhu, *Efficient Algorithms for Finding a Longest Common Increasing Subsequence*, *The 16th Annual International Symposium on Algorithms and Computation,* Hainan, China, 2005, pp.665-674.

[6] M. S. Chang and F. H. Wang, *Efficient Algorithms for the Maximum Weight Clique and Maximum Weight Independent Set Problems on Permutation Graphs*, *Information Processing Letters*, Vol.43, No.6, 1992, pp.293-295.

[7] F. Y. L. Chin, A. D. Santis, A. L. Ferrara, N. L. Ho and S. K. Kim, *A Simple Algorithm for the Constrained Sequence Problems*, *Information Processing Letters*, Vol.90, No.4, 2004, pp.175-179.

[8] A. L. Delcher, S. Kasif, R. D. Fleischmann, J. Peterson, O. White and S. L. Salzberg, *Alignment of Whole Genomes*, *Nucleic Acids Research*, Vol.27, No.11, 1999, pp.2369-2376.

[9] W. L. Hsu, *Maximum Weight Clique Algorithms for Circular-Arc Graphs and Circle Graphs*, *SIAM Journal on Computing*, Vol.14, No.1, 1985, pp.224-231.

[10] J. W. Hunt and T. G. Szymanski, *A Fast Algorithm for Computing Longest Common Subsequences*, *Communications of the ACM*, Vol.20, No.5, 1977, pp.350-353.

[11] I. Katriel and M. Kutz, *A Faster Algorithm for Computing a Longest Common Increasing Subsequence*, Research Report MPI-I-2005-1-007, Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany, March, 2005.

[12] H. Kim, *Finding a Maximum Independent Set in a Permutation Graph*, *Information Processing Letters*, Vol.36, No.1, 1990, pp.19-23.

[13] D. T. Lee and M. Sarrafzadeh, *Maximum Independent Set of a Permutation Graph in k Tracks*, *International Journal of Computational Geometry and Applications*, Vol.3, No.3, 1993, pp.291-304.

[14] D. Liben-Nowell, E. Vee and A. Zhu, *Finding Longest Increasing and Common Subsequences in Streaming Data*, *11th International Computing and Combinatorics Conference,* Kunming, China, 2005, pp.263-272.

[15] E. Mäkinen, *On the Longest Upsequence Problem for Permutations*, Tech. Rep. A-1999-7, Department of Computer Science, University of Tampere, 1999.

[16] F. Malucelli, T. Ottmann and D. Pretolani, *Efficient Labelling Algorithms for the Maximum Noncrossing Matching Problem*, *Discrete Applied Mathematics*, Vol.47, No.2, 1993, pp.175-179.

[17] C. L. Peng, *An Approach for Solving the Constrained Longest Common Subsequence Problem*, Master Thesis, Department of Computer Science and Engineering, National Sun Yat-Sen University, Taiwan, July, 2003.

[18] Y. Sakai, *A Linear Space Algorithm for Computing a Longest Common Increasing Subsequence*, *Information Processing Letters*, Vol.99, No.5, 2006, pp.203-207.

[19] C. Schensted, *Longest Increasing and Decreasing Subsequences*, *Canadian Journal of Mathematics*, Vol.13, 1961, pp.179-191.

[20] Y. T. Tsai, *The Constrained Longest Common Subsequence Problem*, *Information Processing Letters*, Vol.88, 2003, pp.173-176.

[21] C. T. Tseng, S. H. Shiau and C. B. Yang, *An Optimal Algorithm for Finding the Longest Increasing Subsequence of Every Substring*, *Proceeding of the 5th Conference on Information Technology and Applications in Outlying Islands*, 2006, p.14.

[22] P. van Emde Boas, R. Kaas and E. Zijlstra, *Design and Implementation of an Efficient Priority Queue*, *Theory of Computing Systems*, Vol.10, 1976, pp.99-127.

[23] C. B. Yang and R. C. T. Lee, *Systolic Algorithms for the Longest Common Subsequence Problem*, *Journal of the Chinese Institute of Engineers*, Vol.10, No.6, 1987, pp.691-699.

[24] I. H. Yang and Y. C. Chen, *Fast Algorithms for the Constrained Longest Increasing Subsequence Problems*, *Proceeding of the 25th Workshop on Combinatorial Mathematics and Computation Theory*, 2008, pp.226-231.

[25] I. H. Yang, C. P. Huang and K. M. Chao, *A Fast Algorithm for Computing a Longest Common Increasing Subsequence*, *Information Processing Letters*, Vol.93, No.5, 2005, pp.249-253.

[26] M. S. Yu, L. Y. Tseng and S. J. Chang, *Sequential and Parallel Algorithms for the Maximum-Weight Independent Set Problem on Permutation Graphs*, *Independent Processing Letters*, Vol.46, No.1, 1993, pp.7-11.

Biographies

Chiou-Ting Tseng received the B.S. degree and M.S. degree in computer science and engineering from National Sun Yat-sen University, Kaohsiung, Taiwan, in 2003 and 2006, respectively. He is currently a Ph.D. candidate of the Department of Computer Science and Engineering at the National Sun Yat-sen University. His research interests include computer algorithms, bioinformatics and sequence analysis.

Chang-Biau Yang received the B.S. degree in electronic engineering from National Chiao Tung University, Hsinchu, Taiwan, in 1982, and the M.S. degree in computer science from National Tsing Hua University, Hsinchu, Taiwan, in 1984. Then, he received the Ph.D. degree in computer science from National Tsing Hua University in 1988. He is currently a professor in the Department of Computer Science and Engineering, National Sun Yat-sen University. His research interests include computer algorithms, interconnection networks, and bioinformatics.

Hsing-Yen Ann received the B.S. degree and M.S. degree in applied mathematics from National Sun Yat-sen University, Kaohsiung, Taiwan, in 1996 and 1998, respectively. From 1999 to 2004 he joined as an assistant researcher in the Telecommunication Laboratories, Chunghwa Telecom Co., Ltd., Taoyuan, Taiwan. He is currently a Ph.D. candidate of the Department of Computer Science and Engineering at the National Sun Yat-sen University. His research interests include bioinformatics, sequence analysis, pattern matching and software engineering.