Available online at www.sciencedirect.com

ScienceDirect

Information Processing Letters 101 (2007) 55–59

Information Processing Letters

www.elsevier.com/locate/ipl

On the longest increasing subsequence of a circular list [☆]

M.H. Albert [a], M.D. Atkinson [a,*], Doron Nussbaum [b], Jörg-Rüdiger Sack [b], Nicola Santoro [b]

[a] *Department of Computer Science, University of Otago, Dunedin, New Zealand*
[b] *School of Computer Science, Carleton University, 1125 Colonel By Drive, Ottawa, Ontario K1S5B6, Canada*

Received 16 February 2006; received in revised form 1 August 2006; accepted 9 August 2006

Available online 7 September 2006

Communicated by S.E. Hambrusch

Abstract

The longest increasing circular subsequence (LICS) of a list is considered. A Monte Carlo algorithm to compute it is given which has worst case execution time $O(n^{3/2}\log n)$ and storage requirement $O(n)$. It is proved that the expected length $\mu(n)$ of the LICS satisfies $\lim_{n\to\infty}\frac{\mu(n)}{2\sqrt{n}}=1$. Numerical experiments with the algorithm suggest that $|\mu(n)-2\sqrt{n}\,|=O(n^{1/6})$.
© 2006 Elsevier B.V. All rights reserved.

Keywords: Combinatorial problems; Randomized algorithms

1. Introduction

The properties of the longest increasing subsequence (LIS) of a finite sequence of numbers have inspired a number of research areas in mathematics and computer science over many decades. As long ago as 1935 Erdös and Szekeres showed that every sequence of length n has an increasing subsequence or a decreasing subsequence of length about $\sqrt{n}$. It follows immediately that the expected length of an LIS in a random permutation of length n is at least $\frac{1}{2}\sqrt{n}$. That result was the first in a series of investigations (see [3] for a survey) that culminated in the seminal paper [5] which obtained the complete limiting distribution of the length of an LIS in a permutation of length n chosen uniformly at random. An important step in this research was taken by Baer and Brock [4] who correctly estimated the expected length to be $2\sqrt{n}$ by computer simulation. Their work ties in with another aspect of the LIS problem: to find efficient algorithms for computing the LIS. That problem was solved by Schensted [12] by a now classical textbook algorithm (see, e.g., [7,10,11,13]) for computing an LIS in time $O(n\log n)$ based on dynamic programming, and that algorithm in turn has connections to the study of Young tableaux. Computing the LIS has recently gained some practical importance since it is used in the MUMmer system [8] for aligning whole genomes. Fredman [9] has shown that the dynamic programming algorithm is optimal under the comparison model.

In this paper we study a variant of the LIS problem. We shall regard the given sequence as a circular structure. In other words, we shall allow the LIS to wrap around if necessary. The longest increasing circular sequence (LICS) is defined as the longest increasing sub-

[☆] Research supported in part by NSERC and SUN Microsystems.
[*] Corresponding author.
E-mail address: mike@cs.otago.ac.nz (M.D. Atkinson).

sequence when wrap-around is permitted. The LICS is never more than twice as long as the LIS but examples such as $4, 5, 6, 1, 2, 3$ show that this bound can be attained. Throughout the paper we interpret "increasing" in the non-strict sense. Strictly increasing sequences can be found by a minor variation.

Although the LICS problem seems to be a natural extension of the LIS problem we know of only one paper that bears upon it. In [2] a (Las Vegas) randomized algorithm was developed to compute the LIS in all windows of a given span. If applied to a sequence of the form XX (a sequence X concatenated with itself) then, with a window span $n = |X|$, it can compute the LICS of X in worst case time $O(n^2)$ and expected time $O(n^{3/2})$.

The contribution of the present paper is both practical and theoretical. Specifically, we give a Monte Carlo randomized algorithm for the LICS problem whose worst case run-time is $O(n^{3/2} \log n)$ with tiny error probability, and storage requirement $O(n)$ which is simple to program. The algorithm depends on a result (Proposition 1) that gives a connection between the LIS and LICS which appears interesting in its own right. Then we prove that the expected length of the LICS is asymptotic to $2\sqrt{n}$ which, since this is true also for the LIS, is somewhat surprising. Finally we report on some numerical evidence that suggests an even tighter result.

The LICS problem can be thought of as a special case of a class of permutation problems that was introduced in [1]. In this broader framework one is given a fixed set $\mathcal{A}$ of permutations and some input permutation σ of length n; the task is to compute the longest subsequence of σ that is order isomorphic to one of the permutations in $\mathcal{A}$. The ordinary LIS problem is the special case that $\mathcal{A}$ is the set of identity permutations; the LICS problem is the case that $\mathcal{A}$ is the set of all permutations $k+1, k+2, \dots, m, 1, 2, \dots, k$ for some k, m. As noted in [1], there are only few classes $\mathcal{A}$ for which an $O(n \log n)$ algorithm is known.

Besides the fact that circular lists are a natural extension of linear lists another reason for seeking a solution to the LICS problem is that genomes of bacteria are (considered to be) circular [6]; so circular problems do arise naturally.

2. Review of the LIS algorithm

An important ingredient of our approach is the standard dynamic programming algorithm to find an LIS. To make the paper self-contained and to clarify what can be gleaned from this algorithm we review its operation.

Suppose we are given a sequence $X = x_1 x_2 \dots x_n$. We scan the sequence term by term and maintain at every step certain positions $t_1, t_2, \dots, t_r$. The term x_{t_k} is the value of the least possible ending term in an increasing subsequence of length k in the prefix of the sequence that has been scanned to this point. Initially we have $r = 0$ (and $t_0 = 0$, $x_0 = -\infty$, by convention) indicating that, before the sequence is examined, no increasing subsequences have been identified. Notice that we shall necessarily have

$$x_{t_1} \leqslant x_{t_2} \leqslant \cdots \leqslant x_{t_r}$$

since, if $x_{t_{k-1}} > x_{t_k}$, the terminator x_{t_k} of an increasing subsequence of length k will be preceded by the penultimate term of that subsequence, and that term will be a smaller terminator than $x_{t_{k-1}}$ of an increasing subsequence of length $k-1$.

When we inspect the term $y = x_i$ (which we do for values $i = 1, 2, \dots, n$ in turn) we have to update the positions $t_1, t_2, \dots, t_r$. To do this, we locate (using binary search), the index s for which

$$x_{t_{s-1}} \leqslant y < x_{t_s}.$$

If such an index is found then we know that y extends an increasing sequence of length $s-1$ ending at position t_{s-1} and that this new sequence of length s has a smaller terminator than x_{t_s}; thus we redefine t_s to be i. The only situation where s cannot be located is the case $x_{t_r} \leqslant y$; this implies that, for the first time, we have encountered an increasing subsequence of length $r+1$, so we set t_{r+1} equal to i and increment r. We also define back pointers b_i by setting $b_i = t_{s-1}$ (or, in the latter case, $b_i = t_{r-1}$). The back pointers record how x_i was established as the final term of an increasing sequence of length s.

When the entire sequence X has been inspected the final value of r is the length of the LIS. We can then reconstruct (in reverse) the LIS itself by following back pointers from position t_r. Indeed, by recording the value of t_r for each of $i = 1, 2, \dots, n$, we can reconstruct an LIS in any initial segment of X.

Clearly this algorithm takes time $O(n \log n)$. Notice that the algorithm can equally be used to construct a longest decreasing subsequence. It can also operate from right to left if desired.

We note some technical properties of the LIS algorithm. Let $t_1^{(i)}, t_2^{(i)}, \dots$ denote the values of the variables $t_1, t_2, \dots$ at the point that x_i has just been processed. The position i itself will occur among $t_1^{(i)}, t_2^{(i)}, \dots$ and, of course, all the other positions of this set will be less than i. We define s_i by

$$t_{s_i}^{(i)} = i.$$

Thus s_i is the length of the increasing subsequence that x_i created (either for the first time or by having