On Generating and Counting All the Longest Increasing Subsequences

ICHIRO SEMBA*

Suppose that we are given a sequence of n distinct positive integers $1, 2, \ldots, n$. We consider problems generating and counting all the longest increasing subsequences in a given sequence $A_n = a_1 a_2 \ldots a_n$. A generating algorithm is established by using a backtrack technique and requires the running time of max $\{O(n^2), O(l(A_n)m(A_n))\}$, where $l(A_n)$ is the total number of the longest increasing subsequences in A_n and $m(A_n)$ is the length of the longest increasing subsequence in A_n. A counting algorithm is established by using a dynamic programming technique and requires the running time of $O(n^2)$.

1. Introduction

Suppose that we are given a sequence of n distinct positive integers $1, 2, \cdots, n$, denoted by $A_n = a_1 a_2 \cdots a_n$ $(n \geq 1)$. We consider problems generating and counting all the longest increasing subsequences in $A_n = a_1 a_2 \cdots a_n$. These problems are interesting examples of the use of computers in combinatorial mathematics.

A backtrack technique [1] can be used to generate all the longest increasing subsequences in A_n. However, a straightforward application will typically result in an algorithm which is not practical.

In this paper, according to certain rules, decreasing subsequences are constructed from A_n in advance. It requires the running time of $O(n \log_2 m(A_n))$, where $m(A_n)$ is the length of the longest increasing subsequence in A_n. The number of decreasing subsequences is proved to be equal to the length of the longest increasing subsequence in A_n. In Schensted [2], an algorithm determining the length of the longest increasing subsequence in A_n is presented. It requires the running time that is $O(n)$ at the best case and $O(n^2)$ at the worst case. In Fredman [3], an algorithm performing the same task and having a worst case running time of $O(n \log_2 n)$ is described. This bound is shown to be the best possible. Our algorithm for constructing decreasing subsequences from A_n is similar to Fredman's algorithm. Then, a tree representing all the longest increasing subsequences in A_n is constructed from these decreasing subsequences. It requires the running time of $O(n^2)$. Then, in order to generate all the longest increasing subsequences, the backtrack technique is applied to a tree. It requires the running time of $O(l(A_n) m(A_n))$, where $l(A_n)$ is the total number of the longest increasing subsequences in A_n. As a result, the running time for generating all the longest increasing subsequences in A_n is shown to be max $\{O(n^2), O(l(A_n)m(A_n))\}$. A dynamic programming technique [4] is applied to these decreasing subsequences, in order to count all the longest increasing subsequences in A_n. The running time required for counting all the longest increasing subsequences in A_n is proved to be $O(n)$ at the best case and $O(n^2)$ at the worst case.

The longest decreasing subsequence in $a_1 a_2 \cdots a_n$ is the longest increasing subsequence in $a_n \cdots a_2 a_1$. Thus, similar results are obtained for problems generating and counting all the longest decreasing subsequences.

2. Generating Algorithm

In this section, we will establish a generating algorithm for all the longest increasing subsequences in A_n. Our generating algorithm consists of three procedures, Procedure I, Procedure II and Procedure III.

First, we present Procedure I constructing $m(A_n)$ decreasing subsequences $S_1, S_2, \cdots, S_{m(A_n)}$ from A_n in this order.

By Property 4, $m(A_n)$ will be proved to be equal to the length of the longest increasing subsequences in A_n. Procedure I is written in PASCAL-like notation in Fig. 1. We write $S_i < -\phi$ to mean that the subsequence S_i is made empty. We write $S_i < -x$ to mean that the element x is inserted at the end of the subsequence S_i. If the subsequence S_i is empty, then x is put at the head. The last

```
begin
  S0 <- Φ; S0 <- 0; S1 <- Φ; S1 <- a1;
  i:=1;
  for k:=2 to n do begin
    {non-empty subsequences S1,...,Si have been
     constructed in this order}
    if last(Si) < ak then begin
      {construct a new subsequence}
      i:=i+1; Si <- Φ; Si <- ak
    end else begin
      find index j such that last(Sj-1) < ak < last(Sj);
      Sj <- ak
    end
  end
end.
```

Fig. 1 Procedure I constructing subsequences $S_1, \cdots, S_{m(A_n)}$ $(m(A_n) \geq 1)$ from $A_n = a_1 a_2 \cdots a_n$.

*Department of Pure and Applied Sciences, College of General Education, University of Tokyo, Komaba, Meguro-ku, Tokyo 153, Japan.

line number	i	j	k	S_0	S_1	S_2	S_3	S_4
3	1			0	3			
4			2					
6	2			0	3	6		
4			3					
8		2						
9				0	3	64		
4			4					
6	3			0	3	64	9	
4			5					
8	1							
9				0	31	64	9	
4			6					
8		2						
9				0	31	642	9	
4			7					
8		3						
9				0	31	642	95	
4			8					
6	4			0	31	642	95	8
4			9					
8		4						
9				0	31	642	95	87

Fig. 2 The process of Procedure I for $A_9=364912587$.

element of the subsequence S_i is denoted by last (S_i). We will assume that the subsequence S_0 has the element 0. As an example, for $A_9=364912587$, the process of Procedure I is shown in Fig. 2. Four decreasing subsequences $S_1=31$, $S_2=642$, $S_3=95$ and $S_4=87$ are constructed.

Now we will give properties related to Procedure I. The following two properties are obvious.

Property 1 Subsequence $S_i(1\le i\le m(A_n))$ is a decreasing subsequence.

Property 2 Let no subsequences $S_1,\cdots,S_i$ be empty. Subsequences $S_0, S_1,\cdots,S_i$ satisfy that, at line 4 of Procedure I,

$$\text{last}(S_0)<\text{last}(S_1)<\cdots<\text{last}(S_i).$$

Property 3 The set of the subsequences $s_1\cdots s_{m(A_n)}$ such that $s_1\in S_1,\cdots,s_{m(A_n)}\in S_{m(A_n)}$ contains an increasing subsequence of length $m(A_n)$ in A_n.

Proof. Suppose that we are given the element x of a subsequence $S_i(2\le i\le m(A_n))$. By the process of Procedure I, we can find the element y which is contained in S_{i-1} and less than x and appears at the left of x in A_n. Namely, we can construct an increasing subsequence of length $m(A_n)$ in A_n.

Property 4 The quantity $m(A_n)$ is the length of the longest increasing subsequence of A_n.

Proof. Since subsequences $S_1,\cdots,S_{m(A_n)}$ are decreasing subsequences, each of the subsequences $S_1,\cdots,S_{m(A_n)}$ can contain at most one element of any increasing subsequence of A_n. Thus, it follows that the length of any increasing subsequence in A_n is less than or equal to $m(A_n)$.

On the other hand, by Property 3, we can construct an increasing subsequence of length $m(A_n)$. Therefore, $m(A_n)$ is the length of the longest increasing subsequence of A_n.

Secondly, we will present Procedure II constructing a tree which represents all the longest increasing subsequences in A_n. The following property is fundamental to Procedure II.

Property 5 For $1\le i\le m(A_n)$, the ith element of the longest increasing subsequence in A_n is contained in the subsequence S_i and not contained in the subsequences $S_j(j\ne i)$.

Proof. Since each of the decreasing subsequences $S_1,\cdots,S_{m(A_n)}$ can contain at most one element of any longest increasing subsequence in A_n, the ith element of the longest increasing subsequence have to be contained in S_i.

Therefore, the elements of $S_i(1\le i\le m(A_n))$ constitute candidates for the ith element of the longest increasing subsequence in A_n. Suppose that the element x is contained in $S_k(1\le k\le m(A_n))$. In order to construct a tree representing all the longest increasing subsequences in A_n, it is sufficient to determine the set whose elements are contained in S_{k-1} and appear at the left of x in A_n. We denote this set by Son (x) ($1\le x\le n$). If x is the element of S_1 then Son (x) is empty. The element 0 means the root of a tree. Son (0) is defined to be the elements in $S_{m(A_n)}$. We write Son $(x)<-\phi$ to mean that Son (x) is made empty. We write Son $(x)<-y$ to mean that y is inserted in Son (x). We use the following functions in Procedure II.

len (S_k): The length of $S_k(1\le k\le m(A_n))$.
mid (S_k, i): The ith element of $S_k(1\le i\le \text{len}(S_k))$.

```
begin
  for i:=0 to n do Son(i) <- Φ;
  for i:=1 to len(S_m(A_n)) do Son(0) <- mid(S_m(A_n),i);
  for k:=m(A_n) downto 2 do begin
    for j:=1 to len(S_k) do begin
      i:=1;
      while(pos(S_k-1,i)<pos(S_k,j))and(i≤len(S_k-1)) do begin
        if mid(S_k-1,i)<mid(S_k,j) then
          Son(mid(S_k,j)) <- mid(S_k-1,i);
        i:=i+1
      end
    end
  end
end.
```

Fig. 3 Procedure II generating a tree which represents all the longest increasing subsequences in A_n.

pos (S_k, i): The position of the element mid (S_k, i) in A_n.

As an example, for $A_9 = 364912587$, we have shown four decreasing subsequences $S_1 = 31$, $S_2 = 642$, $S_3 = 95$ and $S_4 = 87$, by Procedure I. Thus, we have len $(S_2) = 3$, mid $(S_2, 3) = 2$ and pos $(S_2, 3) = 6$.

If mid $(S_{k-1}, i)(2 \leq k \leq m(A_n))$ is contained in Son (mid (S_k, j)), then mid (S_{k-1}, i) must satisfy the following conditions.

(1) mid $(S_{k-1}, i) <$ mid (S_k, j)

(2) pos $(S_{k-1}, i) <$ pos (S_k, j)

According to these conditions, Procedure II is written in PASCAL-like notation and shown in Fig. 3.

Thirdly, we will present Procedure III generating all the longest increasing subsequences in A_n. Since a tree representing all the longest increasing subsequences in A_n is constructed by Procedure II, it is sufficient to traverse a tree. This tree traversal can be done by using a backtrack technique. Since this technique is well known, Procedure III is omitted.

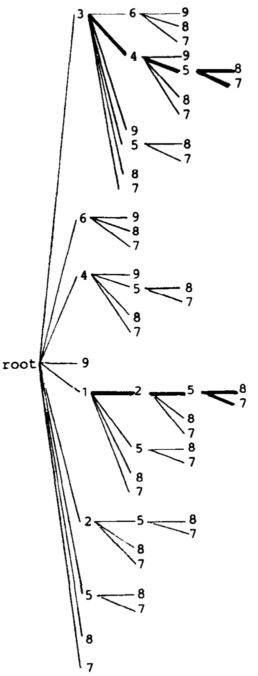

Fig. 4 A tree corresponding to a straightforward backtrack for $A_9 = 364912587$.

x	Son(x)
0	{ 8 , 7 }
1	{ }
2	{ 1 }
3	{ }
4	{ 3 }
5	{ 4 , 2 }
6	{ 3 }
7	{ 5 }
8	{ 5 }
9	{ 6 , 4 }

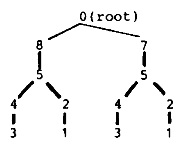

Fig. 5 A tree corresponding to our generating algorithm for $A_9 = 364912587$. Son (x) $(0 \leq x \leq 9)$ are constructed by Procedure II.

As a result, we have the following theorem.

Theorem 1 Combining Procedure I, Procedure II, and Procedure III, all the longest increasing subsequences in $A_n = a_1 a_2 \cdots a_n (n \geq 1)$ are generated.

In order to understand a difference between a straightforward backtrack and our generating algorithm, it is helpful to picture them by using a tree. For $A_9 = 364912587$, the corresponding trees are shown in Figs. 4 and 5. The solutions are indicated by the bold lines. The length of the longest increasing subsequence is 4. There are 4 longest increasing subsequences 3458, 3457, 1258 and 1257.

In a straightforward backtrack, all the increasing subsequences are searched. The length of the longest increasing subsequence can be determined after exhaustive search.

In our generating algorithm, the length of the longest increasing subsequence is determined before search. Candidates for the longest increasing subsequences in A_n are examined and other increasing subsequences are not examined.

3. Analysis of Generating Algorithm

In this section, we will make an analysis of a generating algorithm.

In Procedure I, the number of comparisons is a reasonable measure of the work. Let $F(A_n)$ be the number of comparisons required to construct $m(A_n)$ decreasing subsequences from A_n.

Property 6 For any $A_n (n \geq 1)$,

$$F(A_n) \leq (n-1) + (n - m(A_n))([\log_2 m(A_n)] + 1) \qquad (1)$$

Proof. Since line 5 is executed $n-1$ times, the number of comparisons in line 5 is $n-1$. By Property 2, binary search can be applied to determining the index j in line 8. The number of comparisons required to determine the index j is bounded by $[\log_2 m(A_n)] + 1$. Since line 8 is executed $n - m(A_n)$ times, the number of comparisons in line 8 is $(n - m(A_n))([\log_2 m(A_n)] + 1)$. Comparisons are done in lines 5 and 8. Thus, we obtain (1).

In Procedure II, the number of elements examined to generate a tree is a reasonable measure of the work. Let $G(A_n)$ be the number of elements examined to generate a tree representing all the longest increasing subsequences in A_n.

Property 7 For any $A_n (n \geq 1)$,

$$G(A_n) < n^2. \qquad (2)$$

Proof. If $m(A_n) = 1$, then $G(A_n) = n$. If $m(A_n) = 2$, then $G(A_n) \leq (n+1)^2/4$.

If $m(A_n) > 2$, then it follows that

$$G(A_n) \leq \text{len}(S_{m(A_n)}) + \text{len}(S_1)\,\text{len}(S_2) + \cdots + \text{len}(S_{m(A_n)-1})\,\text{len}(S_{m(A_n)}).$$

By the theorem of the arithmetic and geometric means and the fact that len $(S_1) + \cdots +$ len $(S_{m(A_n)}) = n$, we have

$$\sqrt{\text{len}(S_1)\,\text{len}(S_2)}+\cdots+\sqrt{(\text{len}(S_{m(A_n)-1})+1)\,\text{len}(S_{m(A_n)})}$$
$$\le \text{len}(S_1)/2+\text{len}(S_2)+\cdots+\text{len}(S_{m(A_n)-1})+\text{len}(S_{m(A_n)})/2+1/2$$
$$=n+1/2-(\text{len}(S_1)+\text{len}(S_{m(A_n)}))/2$$
$$<n.$$

Squaring both sides of the above formula, we have (2).

In Procedure III, the number of nodes examined to traverse a tree is a reasonable measure of the work. Let $H(A_n)$ be the number of nodes examined to traverse a tree.

Property 8 For some constant $c>0$,

$$H(A_n)<cl(A_n)m(A_n) \tag{3}$$

Proof. Since the height of the tree is $m(A_n)$ and the number of all the longest increasing subsequences in A_n is $l(A_n)$, the total number of nodes examined by the backtrack technique is bounded by a constant times $l(A_n)$ $m(A_n)$. Thus, we obtain (3).

Theorem 2 The running time required for generating all the longest increasing subsequences in A_n is bounded by

$$\max\{O(n^2),\ O(l(A_n)m(A_n))\}.$$

Proof. By Property 6, 7 and 8, it is easily shown.

4. Counting Algorithm

In this section, we will establish a counting algorithm for all the longest increasing subsequences in A_n. Our algorithm consists of two procedures, Procedure I and Procedure IV. Since Procedure I has been described, we present Procedure IV counting all the longest increasing subsequences in A_n from $m(A_n)$ decreasing subsequences. By Property 5, we can establish Procedure IV by using a dynamic programming technique. Procedure IV is written in PASCAL-like notation and shown in Fig. 6. The integer array $c[k,j]$ denotes the number of increasing subsequences of length k in A_n, whose ith element is contained in $S_i(1\le i\le k-1)$ and kth element is the jth element of S_k.

Thus, the number of all the longest increasing subsequences in A_n is $c[m(A_n),1]+\cdots+c[m(A_n),\text{len}(S_{m(A_n)})]$.

```
begin
  for j:=1 to len(S_1) do c[1,j]:=1;
  for k:=2 to m(A_n) do begin
    for j:=1 to len(S_k) do begin
      c[k,j]:=0; i:=1;
      while(pos(S_k-1,i)<pos(S_k,j))and(i<len(S_k-1)) do begin
        if mid(S_k-1,i)<mid(S_k,j) then c[k,j]:=c[k,j]+c[k-1,i];
        i:=i+1
      end
    end
  end
  record c[m(A_n),1]+...+c[m(A_n),len(S_m(A_n))] as a solution
end.
```

Fig. 6 Procedure IV counting the number of all the longest increasing subsequences in A_n.

Table 1 The values of $c[k,j]$ $(1\le k\le 4,\ 1\le j\le \text{len}(S_k))$ for $A_9=364912587$.

k \ j	1	2	3
1	1	1	
2	1	1	1
3	2	2	
4	2	2	

We obtain the following theorem.

Theorem 3 Combining Procedure I and Procedure IV, the number of all the longest increasing subsequences in A_n is obtained.

As an example, for $A_9=364912587$, we show the values of $c[k,j](1\le k\le 4,\ 1\le j\le \text{len}(S_k))$ in Table 1.

5. Analysis of Counting Algorithm

In this section, we will make an analysis of the counting algorithm.

In Procedure IV, the number of elements examined to be counted is a reasonable measure of the work. Let $I(A_n)$ be the number of elements examined to count all the longest increasing subsequences in A_n.

Property 9 For any $A_n(n\ge 1)$,

$$I(A_n)<n^2. \tag{4}$$

Proof. By Procedure IV, it follows that $I(A_n)\le \text{len}(S_1)$ $\text{len}(S_2)+\cdots+\text{len}(S_{m(A_n)-1})\ \text{len}(S_{m(A_n)})$. Thus, in a similar way as Property 7, we obtain (4).

Theorem 4 The running time required for counting all the longest increasing subsequences in A_n is bounded by $O(n^2)$.

Proof. By Property 6 and 9, it is easily shown.

6. Remarks

For a sequence of n distinct positive integers $B_n=b_1b_2\cdots b_n$, similar results are derived. Let $c_1c_2\cdots c_n$ be a sequence whose element $c_i(1\le i\le n)$ is the order of b_i in $b_1b_2\cdots b_n$. We note that $c_1c_2\cdots c_n$ is a sequence on $\{1, 2, \cdots, n\}$. A sequence $c_1c_2\cdots c_n$ can be determined in $O(n\log_2 n)$ by sorting a sequence $b_1b_2\cdots b_n$. Thus, we may consider a sequence $c_1c_2\cdots c_n$ instead of a sequence $b_1b_2\cdots b_n$.

By Procedure I, $m(B_n)$ decreasing subsequences are obtained. If these decreasing subsequences are merged, then a sequence B_n is sorted in $O(n\log_2 m(B_n))$. This means that an efficient sorting algorithm may be established for a sequence B_n whose $m(B_n)$ is small.

References

1. Bitner, J. R. and Reingold, E. M. Backtrack programming techniques, *Comm. ACM*, 18 (1975), 651–656.
2. Schensted, C. Longest increasing and decreasing subsequences, Canad. *Journal of Math.*, 13 (1961), 179–191.
3. Fredman, M. L. On computing the length of longest increasing subsequences, *Discrete Mathematics*, 11 (1975), 29–35.
4. Bellman, R. Dynamic programming, Princeton University Press, Princeton, N.J., 1957.

(Received March 31, 1982: revised September 7, 1983)