Information Processing Letters 24 (1987) 289–292
North-Holland

16 March 1987

COMPUTING THE NUMBER OF MERGINGS WITH CONSTRAINTS

M.D. ATKINSON and H.W. CHANG

School of Computer Science, Carleton University, Ottawa, Canada K1S 5B6

Communicated by E.C.R. Hehner
Received 9 May 1986
Revised 13 June 1986

A merging of two ordered lists $x_1,\dots,x_p$ and $y_1,\dots,y_q$ is said to be *constrained* if it must satisfy prescribed conditions of the form "x_i precedes y_j" and "y_r precedes x_s". An algorithm is given for calculating the number of mergings which satisfy any given set of constraints. The algorithm has time complexity O(pq) but in many cases runs significantly faster. It is illustrated with examples which involve the Catalan and Fibonacci numbers.

Keywords: Merging, linear extension, Catalan, Fibonacci

AMS Classification: 68C05

1. Introduction

Let $x_1,\dots,x_p$ and $y_1,\dots,y_q$ be two sequences. A *merging* of the two sequences is a sequence $a_1,\dots,a_{p+q}$ where each x_i and y_j occurs exactly once in the sequence, and where $x_1,\dots,x_p$ occur in that order (though not necessarily contiguously) and where $y_1,\dots,y_q$ occur in that order. It is well known that the number of mergings is given by $(p+q)!/p!q!$.

We define a merging to be *C-constrained* if there is a collection C of pairs (x_i, y_j), (y_r, x_s) such that u occurs before v in the merging whenever $(u, v) \in C$. We shall assume that C is *consistent* in the sense that at least one such merging exists.

In this article we consider the problem of calculating the number of C-constrained mergings for an arbitrary collection of constraints C. Knuth gives a formula for the case of a single constraint [2, p. 191]. The general case can be formulated in the language of partially ordered sets.

Define the relation $<$ on $\{x_1,\dots,x_p, y_1,\dots,y_q\}$ by the following conditions:

(a) $x_i < x_{i+1}$, $i = 1, 2,\dots, p-1$,
(b) $y_i < y_{i+1}$, $i = 1, 2,\dots, q-1$,
(c) $u < v$ whenever $(u, v) \in C$,
(d) all transitive consequences of (a), (b), (c).

It is easy to verify that, because C is consistent, the relation is a partial order. The partially ordered set Z has width 2 since it is covered by two chains. Furthermore, the C-constrained mergings are precisely the linear extensions of the poset. Linial [3] mentions a method in which the number of linear extensions is calculated as a determinant whose entries are certain binomial coefficients; if the determinant is evaluated by Gaussian elimination, this method has time complexity $O(n^3)$, where $n = p + q$. In [1], the problem of calculating the number of linear extensions of a poset of width k is solved by an algorithm of time complexity $O(n^{k*k})$ (n^4 if $k = 2$).

We present an algorithm for solving the problem whose time complexity is O(pq). Moreover, the algorithm is a simple one, suitable for hand calculation, and in many cases its complexity is less than pq. We shall give the algorithm, justify it, and illustrate it with two examples which cast an interesting light on the Catalan numbers and Fibonacci numbers.

2. The algorithm

There is an initial preprocessing stage in which the data is organised for input to the computational part of the algorithm. In this first stage we calculate

$$a_i := \min\{t : x_i < y_t\}, \quad i = 1, 2, \ldots, p,$$
$$b_i := \max\{t : y_t < x_i\}, \quad i = 1, 2, \ldots, p.$$

For convenience we invent elements y_0, y_{q+1} which are, respectively, smaller than and greater than every other element in the poset; thus, a_i and b_i are always defined. The time required for preprocessing depends on how the data are presented. For example, a $p \times q$ matrix T might be used, where

$$t_{ij} = \begin{cases} 1 & \text{if } x_i < y_j, \\ -1 & \text{if } y_j < x_i, \\ 0 & \text{otherwise.} \end{cases}$$

Then, O(p log q) time would suffice since the rows of the matrix are monotonic and binary search is possible.

The computational part of the algorithm is as follows:

$(u_0, \ldots, u_q) := (1, 0, 0, \ldots, 0)$
for $i := 1$ **to** p
  sum(u) (1)
  $u_j := 0$ for all $j < b_i$ and all $j \geq a_i$ (2)
endfor
return $\left(\sum_i u_i\right)$

where sum(u) is

for $j := 1$ **to** q
  $u_j := u_j + u_{j-1}$.

We now prove that the algorithm computes the number of C-constrained mergings. Let P_i be the poset induced by poset Z on the elements $x_1, \ldots, x_i$, $y_1, \ldots, y_q$.

Proposition. *At the end of the* i*th iteration of the* **for**-*loop, each* u_j *is the number of linear extensions of* P_i *in which* x_i *lies between* y_j *and* y_{j+1}.

Proof. We use induction on i, and leave the easy verification of the induction base ($i = 1$) to the reader. Suppose now that $i > 1$ and that, at the end of the $(i-1)$st iteration (or the beginning of the i*th* iteration), each u_j is the number of linear extensions of P_{i-1} in which x_{i-1} lies between y_j and y_{j+1}. We shall consider P_i to be constructed from P_{i-1} in two steps.

In the first step, we incorporate the new element x_i and include the order relation $x_{i-1} < x_i$ (and its transitive consequences) to obtain a poset P_i^*. For each j, every extension of P_i^* in which x_i is between y_j and y_{j+1} arises from inserting x_i into an extension of P_{i-1} in which (since x_{i-1} precedes x_i) x_{i-1} must lie between y_k and y_{k+1} for some $k \leq j$. Hence, there are $u_j^* = \sum_{k \leq j} u_k$ extensions of P_i^* in which x_i is between y_j and y_{j+1}. Thus, statement (1) of the algorithm correctly modifies array u so that it contains numbers appropriate for P_i^*.

In the second step, we obtain P_i from P_i^* by imposing relations $y_{b_i} < x_i$, $x_i < y_{a_i}$ and their transitive consequences. These relations are compatible with extensions of P_i^* in which x_i lies between y_{b_i} and y_{a_i} but incompatible with the others. Thus, the number of extensions of P_i^* in which x_i is between y_j and y_{j+1} is u_j^* if $b_i \leq j < a_i$ and is zero otherwise. Thus, statement (2) of the algorithm computes the numbers $u_0, \ldots, u_q$ for P_i.

At the conclusion of the **for**-loop, $P_p = Z$ and hence $u_0 + u_1 + \cdots + u_q$ is the number of linear extensions of Z. □

3. Complexity analysis

It is clear that the algorithm has time complexity pq. There is a significant optimisation which, in favourable cases, sometimes reduces the execution time to O(p) although the worst case is still O(pq). Observe that, on entering the i*th* iteration of the **for**-loop, u_j is nonzero only for $b_{i-1} \leqslant j < a_{i-1}$ (we take $b_0 = 0$) and, on exit, u_j will be nonzero only for $b_i \leqslant j < a_i$. It is sufficient to compute the nonzero values, and the code for the optimised algorithm is

$(u_0, \ldots, u_q) := (1, 0, 0, \ldots, 0)$
for $i := 1$ **to** p
  for $j := b_{i-1} + 1$ **to** $a_i - 1$
    $u_j := u_j + u_{j-1}$
  endfor
endfor
return $\left(\sum_{b_p \leqslant j < a_p} u_j \right)$.

4. Examples

(1) The first example is the poset whose Hasse diagram is given in Fig. 1.

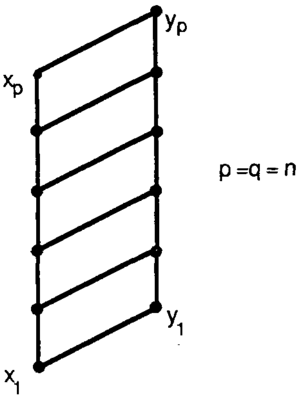

Fig. 1.

It is known that the number of linear extensions of this poset is the *pth* Catalan number c_p. The values in the array u are, in turn,

1						
1	1					
1	2	2				
1	3	5	5			
1	4	9	14	14		
1	5	14	28	42	42	
1	6	20	48	90	132	132

. . . .

where we omit trailing zeros. The algorithm produces each row of this table by partially summing the previous row and duplicating the last element. The row sums (or final elements in each row) give the sequence of Catalan numbers.

A direct proof of this fact can be obtained through the following argument. Consider the tabular array v_{ij}, $i, j = 0, 1, 2, \ldots$ whose first row is $0, -1, -1, -1, \ldots$ and whose first column is $0, 1, 1, 1, \ldots$, and where $v_{ij} = v_{i-1\ j} + v_{i\ j-1}$ if $i, j > 0$. It is easy to see that the strictly lower triangular part of this tableau is the triangular array appearing above. Moreover, standard algebra shows that the ordinary two variable generating function $V(x, y) = \sum v_{ij} x^i y^j$ is $V(x, y) = (x - y)/(1 - x - y)$ which, upon expansion, gives

$$v_{ij} = \binom{i+j-1}{i-1} - \binom{i+j-1}{j-1}.$$

In particular, the numbers $v_{i+1\ i}$ (the diagonal of the triangular array) are the Catalan numbers.

(2) The second example is the poset whose Hasse diagram is given in Fig. 2.

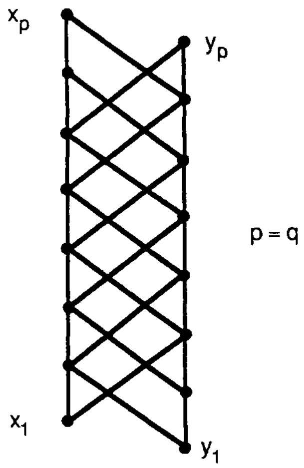

Fig. 2.

This poset is of interest because the complement of its comparability graph is the zigzag poset shown in Fig. 3.

Fig. 3.

The number of linear extensions of the latter poset is known to be the Euler number E_{2p}. The values of the u array as produced by the algorithm are (omitting zeros)

$$\begin{array}{llllll}
1 & 1 & 1 & & & \\
 & 2 & 3 & 3 & & \\
 & & 5 & 8 & 8 & \\
 & & & 13 & 21 & 21 \\
 & & & & \ldots\ldots &
\end{array}$$

The nonzero values r_i, s_i, s_i in its ith row satisfy

$$r_i = r_{i-1} + s_{i-1}, \qquad s_i = r_{i-1} + 2s_{i-1}.$$

It follows that $r_1, s_1, r_2, s_2, r_3, s_3, \ldots$ is the Fibonacci sequence F_0, F_1, $F_2, \ldots$ and that the number of linear extensions of the poset is

$$r_p + s_p + s_p = F_{2p-2} + 2F_{2p-1} = F_{2p+1}.$$

References

[1] M.D. Atkinson and H.W. Chang, Extensions of partial orders of bounded width, Tech. Rept. SCS-TR-83, Dept. of Computer Science, Carleton Univ., 1985.

[2] D.E. Knuth, Sorting and Searching (Addison-Wesley, Reading, MA, 1973).

[3] N. Linial, The information-theoretic bound is good for merging, SIAM J. Comput. 13 (1984) 795–801.