Aplikace matematiky

Juraj Wiedermann

# The complexity of lexicographic sorting and searching

*Aplikace matematiky*, Vol. 26 (1981), No. 6, 432–436

Persistent URL: http://dml.cz/dmlcz/103933

# THE COMPLEXITY OF LEXICOGRAPHIC SORTING AND SEARCHING

JURAJ WIEDERMANN

(Received November 27, 1979)

## 1. INTRODUCTION

Let $U_1, U_2, \ldots, U_k$ be totally ordered sets and let $V$ be a set of $n$ $k$-tuples in the Cartesian product $U_1 \times U_2 \times \ldots \times U_k$. For any $k$-tuple $v$ in $V$, let $c_i(v)$ denote the $i$-th component of $v$.

A lexicographic ordering $\prec$ is defined on $V$ in the usual way, that is, for $v, u \in V$, $v \prec u$ if and only if either $c_1(v) < c_1(u)$ or there exists $1 \leqq j < k$ such that $c_i(v) = c_i(u)$ for $i = 1, 2, \ldots, j$ and $c_{j+1}(v) < c_{j+1}(u)$, where $<$ is the total ordering on each $U_j$.

We shall consider the problem of lexicographic sorting of $k$-tuples of $V$, as well as that of searching for a $k$-tuple in $V$.

The computational complexity of both the problems will be measured by the number of three-branch component comparisons needed for solving these problems (i.e., two components $c_i(v)$ and $c_i(u)$ will be compared yielding $c_i(v) < c_i(u)$, $c_i(v) = c_i(u)$ or $c_i(v) > c_i(u)$ as an answer). We shall be interested in obtaining the (worst case) upper and lower bounds on the complexity, as a function of both $n$ and $k$.

Note that the problem of lexicographic sorting can be straightforwardly solved by applying any "onedimensional" sorting algorithm directly to the $k$-tuples of $V$, which are in this case viewed as "unstructured" elements, with respet to the lexicographic ordering $\prec$. However, this approach would require about $\Theta(n \log n)$ "lexicographic" comparisons, which can need as much as $\Theta(kn \log n)$ component comparisons, because in the worst case the lexicographic order of two $k$-tuples cannot be detected until all $k$ component comparisons are performed.

In a similar manner the lexicographic search can be done in $\Theta(k \log n)$ steps.

In contrast to these "trivial" upper bounds we shall show that making use of the particular structure of the lexicographic ordering, we can accomplish the lexicographic sorting and searching using $\Theta(n(\log n + k))$ and $\lceil \log (n + 1) \rceil + k - 1$ component comparisons, respectively, and that these bounds are asymptotically optimal in the case of sorting and optimal in the case of searching.

In the conclusion of the paper we shall point out some applications of the results, mainly in the databases and also in the theory of sorting.

## 2. LEXICOGRAPHIC SORTING

When it comes to finding a good algorithm for lexicographic sorting of the set $V$ of $n$ $k$-tuples, the divide-and-conquer strategy works well:

**recursive procedure** LEXICOSORT $(S, i)$;
**comment** $S$ is the set of $k$-tuples to be lexicographically sorted; it is supposed that the first $i$-1 components of all $k$-tuples of $S$ are equal, with $1 \leqq i \leqq k$;
**begin**
**if** $|S| = 1$
**then** return $(S)$
**else** let $C$ be the multiset of all $i$-th components of $k$-tuples of $S$;
1. find the median m of the set C;
let $S_1 = \{v \in S \mid c_i(v) < m\}$,
$S_2 = \{v \in S \mid c_i(v) = m\}$,
$S_3 = \{v \in S \mid c_i(v) > m\}$;
2. **if** $|S_1| \neq 0$ **then** return (LEXICOSORT $(S_1, i)$) **fi**;
3. **if** $i = k$ **then** return $(S_2)$
**else** return (LEXICOSORT $(S_2, i + 1)$)
**fi**;
4. **if** $|S_3| \neq 0$ **then** return (LEXICOSORT $(S_3, i)$) **fi**
**fi**
**end**

The procedure is activated by calling LEXICOSORT $(V, 1)$.

The time complexity $T(n, k)$ of LEXICOSORT, applied to the set of $n$ $k$-tuples, is expressed by a recurrence relation

$$T(n, k) \leqq cn + T(n_1, k) + T(n_2, k - 1) + T(n_3, k),$$
$$T(1, k) = T(n, 0) = 0 \quad \text{for} \quad k \geqq 1, \quad n \geqq 1,$$

where the four terms on the right hand side of the relation correspond to the complexity of steps 1 through 4, respectively, in the algorithm LEXICOSORT, with $n_1 = |S_1|$, $n_2 = |S_2|$ and $n_3 = |S_3|$.

Taking into account that $n_1 + n_2 + n_3 = n$, $n_1 \leqq n/2$, $n_3 \leqq n/2$, it is not difficult to verify the solution of the recurrence in the form

$$T(n, k) = \Theta(n(\log n + k)).$$

The example of $n$ $k$-tuples which differ solely in the last component shows that about $\Omega(n(\log n + k))$ comparisons are indeed necessary for lexicographic sorting: we surely need at least $(n-1)(k-1)$ comparisons to detect the equality of the first $k-1$ components of all $n$ $k$-tuples, and it takes $\Omega(n \log n)$ more comparisons to complete the sort with respect to the last components.

Thus we have established the following theorem:

**Theorem 1.** *The lexicographic sort of n k-tuples can be performed by* $\Theta(n(\log n + k))$ *three-branch comparisons.*

We see that if $k = \Omega(\log n)$, the complexity of the lexicographic sort is linear in the size of the input, i.e. in the number of the components of all $k$-tuples.

## 3. LEXICOGRAPHIC SEARCHING

Any lexicographic search algorithm, based on three-branch comparisons, can be viewed as a ternary decision tree. In this tree the components of $k$-tuples are stored at its vertices.

The search for an unknown $k$-tuple $v$ starts in the root by comparing the value of $c_1(v)$ with the value $c_1(r)$ stored in the root. If $c_1(v) < c_1(r)$ ($c_1(v) > c_1(r)$), the search proceeds in a similar way in the left (right) subtree by comparing $c_1(v)$ with the root of this subtree; if $c_1(v) = c_1(r)$, then the value of the first component of $v$ has been found and the search for the next component $c_2(v)$ proceeds now in the middle subtree in an analogous manner.

The search is successful, if all $k$ components of $v$ are found in the tree; otherwise the search ends unsuccessfully.

Consider now the problem of constructing the appropriate lexicographic search tree for a given set $V$ of $n$ $k$-tuples.

There is an alternative way to see the algorithm LEXICOSORT as an algorithm which recursively constructs a lexicographic search tree for the set $S$ of $k$-tuples, whose first $i$-1 components are equal, for $1 \leqq i \leqq k$.

In the root of this tree the value $m$ (found in step 1 of the algorithm) is placed, and the left, middle and right subtree is the lexicographic search tree for the set $S_1$, $S_2$ and $S_3$ of $k$-tuples, in which the first $i-1$, $i$ and $i-1$ components, respectively, are equal (this corresponds to steps 2, 3 and 4, respectively).

We shall call the decision tree, constructed by the algorithm LEXICOSORT, an optimal lexicographic search tree (the reason for this name will become clear later).

When performing the comparisons as dictated by this tree, each unsuccessful comparison $c_i(v) : c_i(r)$ (with an answer '<' or '>') halves the space of the remaining possibilities for $v$ (the inequalities $|S_1| \leqq \lfloor |S|/2 \rfloor$, $|S_3| \leqq \lfloor |S|/2 \rfloor$ always hold). On the other hand, a successful comparison need not decrease the cardinality of the space

of the remaining possibilities for $v$ (if $|S_2| = |S|$), but in any case it makes a step toward the termination of the algorithm by determining the value of one component of $v$ and thus decreasing the "dimensionality" of the remaining search space.

Hence, if $T(n, k)$ denotes the number of three-branch comparisons sufficient to find the $k$-tuple $v$ in the set of $n$ $k$-tuples, it obviously satisfies

$$T(n, k) \leqq 1 + \max\{T(\lceil n/2 \rceil, k),\ T(n, k-1)\},$$
$$T(n, 0) = 0 \quad \text{for} \quad n \geqq 1,$$
$$T(1, k) = k \quad \text{for} \quad k \geqq 1.$$

The solution of this recurrence is given by

$$T(n, k) \leqq \lceil \log(n+1) \rceil + k - 1$$

but again the example of $n$ $k$-tuples with all but the last components equal shows that in the last expression the equality actually holds.

This implies the following theorem:

**Theorem 2.** *The lexicographic search in the set of n k-tuples can be performed in the optimal lexicographic search tree by*

$$\lceil \log(n+1) \rceil + k - 1$$

*three-branch comparisons, and this number is optimal.*

The somewhat weaker from $T(n, k) \leqq \log n + 2k$ of the last result was originally obtained by Freedman [1] and van Leeuwen [3]. Our construction of the optimal lexicographic search tree differs from their construction by a more careful selection of the component around which the set of $k$-tuples is partitioned.

An alternative construction of the optimal lexicographic search tree is also given in [4].

## 4. APPLICATIONS

It is quite natural to view the set $V$ of $n$ $k$-tuples as a file $F$ of $n$ records, each record consisting of $k$ attributes (keys). Then the (lexicographic) search in the set $V$ corresponds to the search for the answer to the exact-match query in the file $F$[2].

Another interpretation of the set $V$ is to understand it as a $k$-ary relation on $U_1 \times U_2 \times \ldots \times U_k$. Given two $k$-ary relations $V$ and $W$ on $U_1 \times U_2 \times \ldots \times U_k$, with $|W| = m$ and $|V| = n$, $m \leqq n$, we can compute their intersection $V \cap W$ as follows: first, we lexicographically sort the set $W$ by $\Theta(m(\log m + k))$ comparisons, and then in the resulting optimal lexicographic search tree we perform $n$ successive searches for elements of $V$; this consumes other $\Theta(n(\log m + k))$ comparisons.

Thus the intersection of two $k$-ary relations $V$ and $W$ can be found by $\Theta(m + k))$ three-branch comparisons.

Theorem 2 can also be used to improve on Fredman's result about the complexity of sorting $x_1, x_2, \ldots, x_n$, provided we know the subset $G$ of all the $n!$ orderings on $x_1, x_2, \ldots, x_n$ to which the resulting ordering belongs [1]. Then, following Fredman (and using Theorem 2) it can be shown that $\lceil \log (|G| + 1) \rceil + n - 1$ comparisons suffice to determine the resulting of $x_1, x_2, \ldots, x_n$ (the original result was $\log |G| + 2n$)

As a further application of this result we can show that if $X$ and $Y$ are $n$-element sets of real numbers, then $n^2 + \Theta(n \log n)$ comparisons suffice to sort the $n^2$-element set $X + Y$, thereby saving a factor of two as compared with the original Fredman's bound $2n^2 + \Theta(n \log n)$ [1]. Moreover, in the same paper Fredman shows that $(n - 1)^2$ comparisons are in fact necessary, so the complexity of sorting $X + Y$ is known with regard to the lower order terms.

*References*


[1] *M. L. Fredman:* How good is the information theory bound in sorting? Theoretical Computer Science 1, 1976, pp. 355—361.

[2] *R. L. Rivest:* Partial-match retrieval algorithms. SIAM J. Computing 5, 1976, pp. 115—174.

[3] *J. van Leeuwen:* The complexity of data organisation. Foundations of computer science II, Part 1. Mathematical centre tracts 81, Mathematisch centrum, Amsterdam 1976.

[4] *J. Wiedermann:* Search trees for associative retrieval (in Slovak). Informačné systémy 1, 1979, pp. 27—41.


## Súhrn

## ZLOŽITOSŤ LEXIKOGRAFICKÉHO TRIEDENIA A VYHĽADÁVANIA


JURAJ WIEDERMANN



V článku je navrhnutý asymptoticky optimálny triediaci algoritmus, ktorý lexikograficky triedi množinu $k$-tic mohutnosti $n$ pomocou $\Theta(n(\log n + k))$ porovnaní medzi jednotlivými komponentami $k$-tic. Na tento lexikografický triediaci algoritmus se dá pozerať aj tak, že v priebehu triedenia vybuduje tzv. optimálny lexikografický vyhľadávací strom, v ktorom sa potom dá optimálne vyhľadávať neznáma $k$-tica pomocou presne $\lceil \log_2 (n + 1) \rceil + k - 1$ porovnaní v najhoršom prípade.



*Author's address:* RNDr. *Juraj Wiedermann*, Výskumné výpočtové stredisko, Dúbravská cesta 3, 885 31 Bratislava.