EFFICIENT SEARCHING USING PARTIAL ORDERING

A. BORODIN [1], L.J. GUIBAS [2], N.A. LYNCH [3] and A.C. YAO [2,4]
[1] *University of Toronto, Toronto, Canada*
[2] *Xerox Palo Alto Research Center, Palo Alto, CA 94304, U.S.A.*
[3] *Georgia Institute of Technology, Atlanta, GA 30332, U.S.A.*
[4] On leave from *Stanford University, Stanford, CA 94305, U.S.A.*

Received 30 October 1979; revised version received 5 December 1980

Algorithm, Dilworth's theorem, membership query, partial order, preprocessing, sorting

0. Introduction

Given a set U of n distinct numbers, we are interested in preprocessing them, so that subsequent membership queries of the form 'Is $y \in U$?' can be answered quickly. For example, if one sorts the elements of U, then each query can be answerred in $\lceil \lg(n+1) \rceil$ comparisons by a binary search. This may be contrasted with the situation where no preprocessing is done, when n comparisons are needed to answer the query. In general, the establishment of a partial order over U will facilitate answering the queries. In this paper we examine the trade-off between the preprocessing cost and the subsequent search cost for each query, in a model using pairwise comparisons among the numbers as the basic operations. Suppose we wish to be able to answer any membership query in at most S(n) comparisons. Can we put a lower bound on P(n), the worst-case cost of a preprocessing algorithm which builds some suitable partial orders on U? We shall show that $P(n) + n \lg S(n) \geqslant (1 + o(1))n \lg n$ for any comparison-based algorithm [5]. This result can be extended in a straightforward manner to the case where some numbers in U may be identical. A simple constructive argument will also show that it is best possible.

For some related work, see the papers of Detig et al. [1], and Munro and Suwanda [6].

[5] The notation lg denotes log to the base 2.

1. The model

Let $x_1, x_2, \ldots, x_n$ denote the numbers [6] in the U. Initially we know nothing about the relative ing of the x_i. We will use a preprocessing algori discover some of these ordering relations. This mation will then be made available to a search a rithm. Precisely speaking, a *preprocessing algorit* is a 'comparison tree' (see [4]), with each intern node containing a comparison of the form $x : x'$ ($x, x' \in U$). Algorithm *P* works by tracing a path from the root down, making at each internal node the spe fied comparison and choosing an exit link according to the result ($<$, or $>$). At each step, the results of all previous comparisons define a partial order *U* on U, which is, of course, consistent with the underlying linear order of the x_i. A leaf in the comparison tree indicates termination of the preprocessing algorithm. Accordingly, associated with each leaf v we have a partial order U_v on U. These partial orders will necessarily be distinct, but may often fall into a few isomorphism classes.

Given a partial order *U*, we will be interested in *search algorithms* that answer correctly the query 'Is $y \in U$?' for any number y and any set of n distinct numbers U satisfying *U*. Such algorithms can also be represented by comparison trees, with each internal node containing a comparison of the form $x : x'$

[6] Although we speak of 'numbers', clearly the same considerations apply to elements from any totally ordered set.

$(x, x' \in U \cup \{y\})$. These comparisons now have ternary outcomes ($<$, $=$, or $>$). When a leaf is reached, the ordering information the algorithm has discovered, along with that implicit in U, must determine if y is in U. Note that we allow the search algorithm to perform comparisons among elements in U. As we will see, our upper-bound results will remain true even under the condition that all comparisons of the search algorithm involve y.

We must generalize this concept of a search algorithm slightly, to deal with the fact that a preprocessing algorithm can generate several different partial orders. For us, a search algorithm S will denote a table of algorithms like the above, indexed by some collection of partial orders. Thus the inputs to S are both y and the (index of the) partial order from the collection to be searched. We will call a pair (P, S) of preprocessing and search algorithms *compatible*, if the collection of partial orders S can search includes all those P can generate.

We now associate worst-case cost measures with the algorithms. The cost of P, denoted as $c(P)$, is the maximum number of comparisons made during preprocessing, i.e. the height of the corresponding comparison tree. Similarly, the cost of a search algorithm S, denoted by $c(S)$, is the maximum number of comparisons made during a search, taken over all partial orders in the algorithm's collection, and for each partial order over the set of all possible paths in the comparison tree.

Note that comparisons are counted as contributing to preprocessing or search according to our choice in drawing the boundary between the algorithms. For instance, instead of sorting during preprocessing and then using binary search, we could run the sort only up to an intermediate point, and then complete it as part of the search phase, before the final binary search. We only insist that (1) all comparisons involving the new element y belong to the search phase, and (2) that all preprocessing comparisons precede any search comparison.

Our main result is the following trade-off theorem between the two costs.

Theorem 1. Let (P, S) denote a pair of compatible preprocessing and search algorithms on a set of n numbers U. If $c(P)$ denotes the preprocessing cost of P and $c(S)$ denotes the search cost of S, both measured in comparisons, then

$$c(P) + n \lg c(S) \geqslant (1 + o(1))n \lg n.$$

Furthermore, this bound is best possible.

This of course implies that the worst-case preprocessing cost P(n) of any algorithm with a compatible seach algorithm of worst-case cost not exceeding S(n) satisfies the inequality

$$P(n) + n \lg S(n) \geqslant (1 + o(1))n \lg n.$$

This covers both the cases $P(n) = 0$, $S(n) = n$, and $P(n) = n \lg n$, $S(n) = \lceil \lg(n + 1) \rceil$ mentioned in the Introduction.

2. Upper bounds

In this section we show that the result of Theorem 1 is best possible by exhibiting a simple partial order U and a pair of compatible algorithms (P, S) where P always generates (partial orders isomorphic to) U, S has a prescribed search cost $s = c(S) \geqslant \lceil \lg(n + 1) \rceil$ and for which

$$c(P) + n \lg c(S) \leqslant (1 + o(1))n \lg n. \qquad (1)$$

We make use of disjoint sorted lists of lengths as equal as possible. Suppose we use k lists, a of length $m = \lceil n/k \rceil$, and b of length $m - 1$, a, b defined by $am + b(m - 1) = n$ and $a + b = k$. In the worst-case, we must search each of the k lists using binary search, for a total cost of

$$s = a\lceil \lg(m + 1) \rceil + b\lceil \lg m \rceil.$$

(An appropriate value of k can be computed from this relation, as long as $s \geqslant \lceil \lg(n + 1) \rceil$.)

For preprocessing we need to form the sorted lists, at a total cost of

$$c(P) = am\lceil \lg m \rceil + b(m - 1)\lceil \lg(m - 1) \rceil$$

comparisons in the worst case.

Note that

$$c(S) = s \leqslant k\lceil \lg(m + 1) \rceil,$$

and so

$$\lg c(S) \leqslant \lg k + \lg\lceil \lg(m + 1) \rceil.$$

Similarly

$$c(P) \leqslant km\lceil \lg m \rceil.$$

Using the fact that $\lg\lceil \lg(m+1) \rceil = o(\lg n)$, we finally conclude

$$c(P) + n \lg c(S) \leqslant km\lceil \lg m \rceil + n \lg k + n\, o(\lg n)$$
$$= (1 + o(1))n \lg n$$

which gives (1).

Note that this construction attains the bound under fairly restricted conditions. Firstly, all partial orders produced by the preprocessing algorithm are isomorphic, and secondly, all comparisons of the search algorithm involve the new element y.

3. Lower bounds

We now prove that $P(n) + n \lg S(n) \geqslant (1 + o(1))n \lg n$. First we define some terminology. Let U be a partial order on a set U. We shall write $x <_U x'$ to denote that 'x is less than x' in U'. Two distinct elements x, x', are *U-incomparable* if neither $x <_U x'$ nor $x' <_U x$. A subset $U' \subseteq U$ is incomparable if every two distinct elements in U' are. A *U-chain* is a sequence $z_1 <_U z_2 <_U \cdots <_U z_t$. A family of *U*-chains *partitions* U if every element of U appears in exactly one of the chains. The *width* of U is defined as

$$w(U) = \{\max |U'| \mid U' \subseteq U \text{ is } U\text{-incomparable}\}.$$

We need the following result due to Dilworth [3].

Theorem (Dilworth). Let U denote a partial order on U. Then

$$w(U) = \min\{k \mid \exists \text{ a family of k } U\text{-chains that partitions U}\}.$$

Let $S(U)$ denote the minimum of $S(S)$, taken over all algorithms S capable of searching U.

Lemma 1. If U is empty, then $S(U) \geqslant n$.

Proof. Let S be any search algorithm for U, and $y \notin U$ be any number. Consider any sequence σ of comparisons made by S. Construct an undirected graph G with vertex set $V = U \cup \{y\}$ and edge set $E = \{\{x, x'\} \mid x : x' \text{ is in } \sigma\}$. It is easy to see that G must be connected, otherwise y can be equal to any element in a component different from the one searched, contradicting that $y \notin U$. Thus, there must be at least n edges in E, meaning that S makes at least n comparisons. This proves $S(S) \geqslant n$.

Lemma 2. We have $S(U) \geqslant w(U)$.

Proof. Let $U' \subseteq U$ be any incomparable subset with $|U'| = w(U)$, and U_1, U_2 be the sets $\{x \mid x <_U x' \text{ for some } x' \in U'\}$, $\{x \mid x' <_U x \text{ for some } x' \in U'\}$, respectively. Since U' is incomparable, U_1, U_2, U' are pairwise disjoint.

Let S be any search algorithm on U using U. Let U' be a partial order on U consistent with U, and such that $x <_{U'} x' <_{U'} x''$ for any $x \in U_1, x' \in U', x'' \in U_2$. If U is known to satisfy U' and if y is known to satisfy $U_1 < y < U_2$, then all comparisons except those between elements in $U' \cup \{y\}$ are not needed to decide if $y \in U$. In fact, if we delete from S every internal node v with its comparisons not between elements in $U' \cup \{y\}$, and connect any son of v directly to the father node of v, the resulting tree S' still decides if $y \in U$. Since $y \in U$ if and only if $y \in U'$ under the present conditions, S' is a search algorithm to decide if $y \in U'$ with empty partial order. By Lemma 1, $c(S') \geqslant |U'| = w(U)$. Hence $S(U) \geqslant S(U') \geqslant w(U)$.

Lemma 3. Let $w \geqslant w(U)$, $m = \lceil n/w \rceil$, and integers a, b be defined by $am + b(m-1) = n$ and $a + b = w$. Then $\nu(U)$, the number of linear orderings on U consistent with U is at most

$$\frac{n!}{(m!)^a((m-1)!)^b}.$$

Proof. By Dilworth's theorem, U can be partitioned into w disjoint U-chains of lengths, say, $\ell_1, \ell_2, \ldots, \ell_w$ (with $\Sigma \ell_i = n$). Clearly

$$\nu(U) \leqslant \binom{n}{\ell_1, \ell_2, \ldots, \ell_w}.$$

The lemma follows as the multinominal coefficient achieves maximum when a of the ℓ_i are m and the rest are $m - 1$.

Lemma 4. Let P be any preprocessing algorithm on U which produces partial orders that can be searched in at most s comparisons. Then $c(P) \geq a \lg(m!) + b \lg((m-1)!)$, where $m = \lceil n/s \rceil$, and a, b are defined by $am + b(m-1) = n$ and $a + b = s$.

Proof. At each leaf v, algorithm P must produce a partial order Q_v such that $S(Q_v) \leq s$, and hence $w(Q_v) \leq s$ by Lemma 2. According to Lemma 3, this implies

$$\nu(Q_v) \leq \frac{n!}{(m!)^a ((m-1)!)^b},$$

where m, a, b are as defined. Since all n! linear orderings on U must end in some leaf, we have

$$n! \leq 2^{c(P)} \frac{n!}{(m!)^a ((m-1)!)^b}.$$

This leads to $2^{c(P)} \geq (m!)^a((m-1)!)^b$. Hence the lemma.

Note that the same conclusion a fortiori holds if we restrict the comparisons of the search algorithm to those involving y only.

Proof of Theorem 1. From Lemma 4 we have

$$c(P) \geq s \lg((m-1)!)$$
$$\geq s((n-1)\lg(m/e))$$
$$\geq (n-s)\lg(n/se),$$

where we have used the easily proved inequality $q! \geq ((q+1)/e)^q$. The above inequality can be rewritten as

$$c(P) + n \lg s \geq n \lg n - n \lg e - s \lg(n/se).$$

Since $\lceil \lg(n+1) \rceil \leq s \leq n$ the last term on the right-hand side is O(n), and so we have shown from which Theorem 1 follows.

$$c(P) + n \lg s \geq (1 + o(1))n \lg n,$$

It may be of interest to observe that a somewhat weaker version of the same theorem can be obtained by an adversary argument. The adversary keeps markers 1, 2, ..., n in *bins* to help it organize its response strategy while the preprocessing algorithm runs. Bins are organized into an infinite binary tree, with all markers initially at the root. When the algorithm asks about $x_i : x_j$, the adversary examines the current positions of i and j in the bin tree.

Case 1. If i is in a bin to the left (right) of j's bin and these bins are not on a common path, then the adversary answers $<$ ($>$).

Case 2. If i is in a bin which is a proper ancestor of j's bin (or vice versa), then the adversary moves i to whichever son is not an ancestor of j and answers $<$ ($>$), if i is to the left (right) of j (symmetrically for i and j interchanged).

Case 3. If i and j are in the same bin, then the adversary moves i to the left son, j to the right son, and answers $<$.

In the analysis of the adversary, we give a lower bound for the number of markers which must lie on some common path in the final bin tree. Since the corresponding elements are all incomparable in the final partial order, Lemma 2 can be applied to yield a lower bound on the search time. The resulting inequality is $2P(n) + n \lg S(n) \geq (1 + o(1))n \lg n$.

4. Remarks and open problems

The results of this paper extend readily to the case where equal keys may be present. One need only check that the upper-bound construction is still valid.

There is a discrepancy between our lower and upper bounds in the *second-order* term. The upper bound gives a second-order term of order O(n lg lg n), while the lower bound has a *negative* such term of order O(n). It may be of interest to further close this gap.

We conjecture that a similar trade-off between preprocessing and search costs holds under the average-cost metric as well. However, we have not been able to relate the average search cost of a partial order to the number of permutations consistent with it.

Finally, we can try to generalize these results to 'on-line' algorithms. In [1] an on-line data-structure, called *binomial lists*, is given with preprocessing cost O(n lg n) and search cost $O(\lg^2 n)$ in the worst-case. It is also shown there that this is essentially best-possible, if one considers only partial orders consisting of disjoint sorted lists. But without this restriction, no 'on-line' lower bounds are known.

References

[1] J. Bentley, D. Detig, L. Guibas and J. Saxe, An optimal data structure for minimal-storage dynamic member searching, CMU (1978), unpublished manuscript.

[2] M. Blum, R.W. Floyd, V. Pratt, R. Rivest and R. Tarjan, Time bounds for selection, J. Comput. System Sci. 7 (1973) 448–461.

[3] R.P. Dilworth, A decomposition theorem for partially ordered sets, Ann. of Math. 51 (1950) 161–166.

[4] D.E. Knuth, The Art of Computer Programming, Vol. 3 (Addison-Wesley, Reading, MA, 1973).

[5] A. Schönhage, M.S. Paterson and N. Pippenger, Finding the Median, J. Comput. System Sci. 13 (1976) 184–199.

[6] J.I. Munro and H. Suwanda, Implicit data structures, Proc. 11th Annual STOC Symposium, Atlanta, GA, 1979, 109–117.