GENERAL METHODS FOR "ALL ELEMENTS" AND "ALL PAIRS" PROBLEMS

Mark H. Overmars

RUU-CS-80-7
September 1980

Rijksuniversiteit Utrecht

Vakgroep informatica

Princetonplein 5
Postbus 80.002
3508 TA Utrecht
Telefoon 030–531454
The Netherlands

GENERAL METHODS FOR "ALL ELEMENTS" AND "ALL PAIRS" PROBLEMS

Mark H. Overmars

Technical Report RUU-CS-80-7

September 1980

Department of Computer Science
University of Utrecht
P.O. Box 80.002, 3508 TA Utrecht
the Netherlands

GENERAL METHODS FOR "ALL ELEMENTS" AND "ALL PAIRS" PROBLEMS*

Mark H. Overmars

Department of Computer Science, University of Utrecht
P.O. Box 80.002, 3508 TA Utrecht, the Netherlands

Abstract. Two methods are given for transforming a static data structure for a search problem into a structure for answering the corresponding "all elements" or "all pairs" problem. The first method is applicable to search problems that are decomposable. The second one merely puts a restriction on the given static data structure for the search problem.

Keywords and phrases. All elements problem, all pairs problem, decomposable searching problem.

1. Introduction.

Searching problems received much attention during the last few years.

Definition 1.1. Given a set A of elements of type T_1 and an object p of type T_2 a searching problem PR(A,p) maps A and p to an answer of type T_3.

A first question is how to represent A as a static data structure such that queries PR(A,p) can be answered efficiently. An often more difficult question is how to represent A as a dynamic structure, such that we can insert and delete points in A. Bentley [1] and many others after him have given standard methods to transform a static structure for A into a dynamic structure under suitable assumptions on PR.

In this paper we will consider a different transformation for special classes of searching problems. When the elements of A are of the same type as the objects we perform queries with (T_1=T_2) we can describe a corresponding "all elements" problem.

---

* This work was supported by the Netherlands Organization for the Advancement of Pure Research (Z.W.O.).

Definition 1.2. Given a searching problem PR with $T_1=T_2$, the all elements problem $PR_e(A)$ asks for all ordered pairs $(p, PR(A/\{p\},p))$ with $p \in A$.

Hence we want to perform a query with each point $p \in A$ on the rest of A. Consider, for example, the nearest neighbour searching problem ([3]): Given a set A of points in the plane and a point p, determine a point in A with minimal distance to p. This clearly is a searching problem with $T_1=T_2$. The corresponding "all elements" problem, the so-called all nearest neighbours problem, asks for each point $p \in A$ its nearest neighbour in $A/\{p\}$.

For some problems we can reformulate definition 1.2. in a somewhat stronger sense. This is the case when the answer to a query PR(A,p) consists of some subset A' of A and $q \in A'$ only if there is some specific relation between p and q (which we assume to be reflexive). In this case we only ask for the pairs (p,q) satisfying the relation. It is the so-called "all pairs" problem.

Definition 1.3. Given a searching problem PR satisfying $T_1=T_2$, $T_3=2^{T_1}$ and $q \in PR(A,p) \Leftrightarrow q \in A \wedge R(p,q)$ for some reflexive relation R, the all pairs problem $PR_p(A)$ asks for all pairs (p,q) with $p,q \in A$ and R(p,q).

For an "all pairs" problem we do not want (p,q) and (q,p) to be reported both. Consider, for example, the rectangle intersection searching problem ([2]): Given a set A of rectilinearly oriented hyper rectangles in d-dimensional space and another such rectangle p, determine all elements in A that intersect with p. Clearly, the problem has the desired properties. The corresponding all pairs problem asks for all pairs of rectangles in A that intersect (duplicates not allowed).

In Section 2 we show how to transform a data structure for a searching problem into a structure to answer the corresponding all elements or all pairs problem. It requires that we have a structure for PR that permits us to perform insertions in some specific order (in average or worst-case time). Bentley [1] has shown that for a special subclass of searching problems, the so-called decomposable searching problems, we can transform a static structure into a structure that permits us to perform insertions efficiently.

Definition 1.4. A searching problem PR is called decomposable if and only if

$$\forall A,B \in 2^{T_1}, \forall p \in T_2 \quad PR(A \cup B,p) = \square\,(PR(A,p), PR(B,p))$$

for some efficiently computable operator $\square$.

It follows that for decomposable searching problems PR we can solve PR_p. In Section 3 we show that for decomposable searching problems we can solve PR_e as well.

A second transformation is based on deletions rather than insertions.

Definition 1.5. A searching problem PR, together with a structure S for it, is called a D_O-searching problem if and only if S allows us to delete points (or to replace points by a kind of dummy object that does not affect later queries) in a specific order ORD, which does not increase the future query or deletion time. Let $D_S(n)$ be the time needed for such a "deletion" in a structure of n points.

We show that for D_O-searching problem we can solve PR_p efficiently. When S allows us to perform arbitrary deletions (rather than in order) we can solve PR_e too (in this case we call PR a D-searching problem).

To express time bounds we need the following

Definition 1.6. Let S be a structure for a searching problem PR containing n points.

$Q_S(n)$ = the time required to perform a query on S.
$P_S(n)$ = the time required to build S from n points.
$S_S(n)$ = the space needed for S.
$I_S(n)$ = the average time required to perform an insertion on S (when applicable).
$T_e(n)$ = the time needed to solve an all elements problem of n points.
$T_p(n)$ = the time needed to solve an all pairs problem of n points.

We assume that Q_S and I_S are nondecreasing and that P_S and S_S are at least linear.

2. The all pairs problem.

We consider the all pairs problem first. Let $A = \{a_1, \ldots, a_n\}$. We can solve the all pairs problem as follows. First check a_2 with respect to a_1, then check a_3 with respect to $\{a_1, a_2\}$ etc. In general, check a_i with respect to $\{a_1, \ldots, a_{i-1}\}$. In this way each pair is checked exactly once. To check a_i w.r.t. $\{a_1, \ldots, a_{i-1}\}$, we need that $\{a_1, \ldots, a_{i-1}\}$ is available as a structure S and that we can perform a query on S. When S permits us to perform insertions in a specific order ORD we first sort A with respect to ORD. We start inserting a_1 into a empty S and perform a query with a_2. Then we insert a_2 in S and we perform a query with a_3. In general at the moment S consists of $\{a_1, \ldots, a_{i-1}\}$ we perform a query with a_i and insert a_i afterwards.

Theorem 2.1. Let PR be a searching problem for which there exists a corresponding all pairs problem. Let S be a data structure for PR that permits us to perform insertions in a specific order. Then we can solve PR_p in time $T_p(n) \leq n(Q_S(n) + I_S(n)) + ORD(n)$, where ORD(n) denotes the time needed to sort the points according to the ordering.

Proof

First we have to sort the set which takes ORD(n). Next we perform with each point one query and one insertion on a structure that contains less than n points. The bound follows.

□

Note that the storage required remains $S_S(n)$. One easily verifies that it does not matter whether I_S is an average or a worst-case bound. A well-known class of searching problem to which this method applies is the class of decomposable searching problems. Bentley [1] showed that once we have a static structure S for a decomposable searching problem PR we can transform it into a structure D in which we can perform insertions. The bounds for D are

$$Q_D(n) = O(\log n)\, Q_S(n)$$
$$I_D(n) = O(\log n)\, P_S(n)/n$$
$$S_D(n) = S_S(n)$$

It follows that for a decomposable searching problem we can solve PR_p in $T_p(n) \leq O(n \log n)(Q_S(n) + P_S(n)/n)$, while the required storage does not increase. When $Q_S(n) \neq O(P_S(n)/n)$ one can get better results by applying methods of Overmars and van Leeuwen [4].

Another class of searching problems for which we can solve the corresponding all pairs problem is the class of D_O-searching problem. In this case we work the other way round. We first build a structure of all the points. Then we delete the points one by one according to the ordering and perform a query with the deleted point on the structure that remains. One easily sees that in this way each point is checked exactly once with respect to each other point.

Theorem 2.2. Given a D_O-searching problem PR that satisfies the conditions for all pairs problems. Let S be a data structure for PR. Then we can solve PR_p in $T_p(n) \leq P_S(n) + n.(Q_S(n) + D_S(n)) + ORD(n)$.

Proof

We first build S, which takes $P_S(n)$. Then we order the points, which takes ORD(n). For each point we perform one query and one deletion. Because the deletion does not increase the future deletion or query time this takes $(Q_D(n) + D_S(n))$ per point. The bound follows.

□

Assuming that a deletion does not increase the storage needed, we still need no more than $S_S(n)$ space. For example, the all intersecting rectilinearly oriented line segments problem asks for all pairs of rectilinearly line segments in a set A that intersect. Vaishnavi and Wood [7] describe a structure for answering the corresponding search problem in O(log n). Their structure can be built in O(n log n). It satisfies the condition for D_O-searching problems with $D_S(n) = O(\log n)$. Hence we can solve the corresponding all pairs problem in O(n log n). (Of course we have to add tc this bound the number of reported pairs.)

3. The all elements problem.

Let us now consider the all elements problem. Let PR be a decomposable searching problem with $T_1 = T_2$. To solve the corresponding all elements problem we first build a structure D (in the sense of Bentley) by inserting the points one by one. When we keep a record of how we did this (in what order), we can again undo these operations and delete the points one by one in opposite order. These points we collect again by inserting them in another dynamic structure D'. So after building D we delete the points one after the other, perform a query with the deleted point on both D and D' and insert it in D' afterwards. In this way we perform with each point $p \in A$ a query on the entire set $A \setminus \{p\}$.

Theorem 2.3. Given a decomposable searching problem with a static structure S, we can answer the corresponding all elements problem in

$$T_e(n) \leq O(n \log n)(Q_S(n) + P_S(n)/n)$$

Proof

By the result of Bentley [1] we can transform S into a structure D with $Q_D(n) = O(\log n)\ Q_S(n)$ and $I_D(n) = O(\log n)\ P_S(n)/n$. We first have to build D by n insertions which takes $O(n \log n)\ P_S(n)/n$. Next we have to perform with each point a deletion, two queries and an insertion. The deletion time will be of the same order as the insertion time (because we reverse the process), hence for each point we have to do $O(\log n)(Q_S(n) + P_S(n)/n)$ work. The bound follows.

□

Again the storage needed remains $O(S_S(n))$.

We can also solve the all elements problem for D-searching problems. In this case we first build the structure S out of the points. Then for each point we delete it, perform a query with it and reinsert it again. This reinsertion can be performed when we remember how we performed the deletion and will also take at most $O(D_S(n))$.

Theorem 2.4. Given a D-searching problem PR with structure S, we can solve PR_e in $T_e(n) \leq O(n)(Q_S(n) + D_S(n)) + P_S(n)$.

Proof

Building S takes $P_S(n)$. Next we have to perform with each point one deletion, one query and one reinsertion. This takes no more than $O(D_S(n) + Q_S(n))$. The bound follows.

□

The amount of storage required remains unaffected.
Consider, for example, the following searching problem: Given a set A of points in the plane and another point p, determine the tangents through p to the convex hull of A (if any). The corresponding all elements problem can be formulated as follows: Given a set of points A, determine for each point what changes it caused on the convex hull. Using a structure of Overmars and van Leeuwen [5, 6] we can answer the search problem in $Q_S(n) = O(\log n)$ while $P_S(n) = O(n \log n)$. The structure satisfies the condition of D-problems with $D_S(n) = O(\log^2 n)$. Hence we can solve the all elements problem in $O(n \log^2 n)$.

3. References.

[1] Bentley, J.L., Decomposable searching problems, Inform. Proc. Lett. 8 (1979) pp. 244-251.

[2] Edelsbrunner, H., Dynamic rectangle intersection searching, Techn. Rep. 47, Inst. für Informatik, Technische Universität Graz, 1980.

[3] Kirkpatrick, D.G., Optimal search in planar subdivisions, Dept. of Computer Science, UBC, Vancouver, Canada (1979).

[4] Overmars, M.H. and J. van Leeuwen, Some principles for dynamizing decomposable searching problems, Techn. Rep. RUU-CS-80-1, Dept. of Computer Science, University of Utrecht, 1980. (To appear in Inform. Proc. Lett.)

[5] Overmars, M.H. and J. van Leeuwen, Maintenance of configurations in the plane, Techn. Rep. RUU-CS-79-9, Dept. of Computer Science, University of Utrecht, 1979/1980.

[6] Overmars, M.H. and J. van Leeuwen, Notes on maintenance of configurations in the plane, Techn. Rep. RUU-CS-80-5, Dept. of Computer Science, University of Utrecht, 1980.

[7] Vaishnavi, V.K. and D. Wood, Rectilinear line segment intersection, layered segment trees and dynamization, Techn. Rep. 80-CS-8, Unit for Computer Science, Mc Master University, Hamilton, Canada, 1980.