Information Processing Letters 25 (1987) 269–273
North-Holland

17 June 1987

A log log n DATA STRUCTURE FOR THREE-SIDED RANGE QUERIES

O. FRIES, K. MEHLHORN, S. NÄHER and A. TSAKALIDIS

Fachbereich 10 — Angewandte Mathematik und Informatik, Universität des Saarlandes, D-6600 Saarbrücken 11, Fed. Rep. Germany

Communicated by T. Lengauer
Received 17 September 1986

We present a data structure for three-sided range queries for a set of n points in $\mathbb{R}^2$ where the x-coordinates of the points are restricted to the range $\{1, 2, \ldots, N\}$. The answer to a three-sided range query is the set of points lying in the semi-finite strip defined by three coordinates x_0, x_1, and y_0. The data structure is based on the priority search tree of McCreight (1981). The basic idea is to start the search at the leaves of the tree and to reduce the query to a constant number of three-sided range queries on sets of O(log n) points. For this purpose we extend the basic priority search tree T by adding four secondary priority search trees to each search path in T. This extended priority search tree can answer three-sided range queries in $O(\log\log n + k)$ time and $O(N + n)$ space (here, k is the cardinality of the answer).

Keywords: Priority search tree, range search, lowest common ancestor, heap

1. Introduction

Let S be a set of n points in $[1..N] \times \mathbb{R}$. A three-sided range query is given by three coordinates $x_0, x_1 \in [1..N]$ and $y_0 \in \mathbb{R}$. The output of the query are all points in S whose first coordinate lies between x_0 and x_1 and whose second coordinate is not greater than y_0, i.e., we have to compute the set

$$L = \{(x, y) \in S \mid x_0 \leqslant x \leqslant x_1 \text{ and } y \leqslant y_0\}.$$

The *priority search tree* [3] is a well-known data structure supporting such three-sided range queries (sometimes also called $1\frac{1}{2}$-dimensional range queries) in the two-dimensional plane $\mathbb{R}^2$ in $O(\log n + |L|)$ time and O(n) space. It also supports insertion and deletion of points in O(log n) time. A detailed description of the priority search trees and its applications can be found in [2].

In the present paper we assume that the plane is bounded in one dimension. More precisely, the x-coordinates of all points in S from the finite set [1..N]. This restriction enables us to present a data structure that answers three-sided range queries in $O(\log\log n + |L|)$ time, needs $O(N + n)$ space and can be build in $O(N + n)$ time. We want to point out that our result is different from the result of Overmars [4]. He presents a data structure for range searching on a two-dimensional bounded grid $[1..N] \times [1..N]$ with $O(\log\log N + |L|)$ query time and O(n log n) space.

In Section 2 we give a short description of the priority search tree as introduced by McCreight. In Section 3 we show how to extend the priority search tree in order to achieve double logarithmic searching time in $O(N + n \log n)$ space. In Section 4 we reduce the space requirement to $O(N + n)$ and show how to construct the data structure in $O(N + n)$ time.

2. The priority search tree

Let S be a set of n points in $[1..N] \times \mathbb{R}$ whose y- and x-coordinates are pairwise distinct (this re-

striction will be dropped later) and let

$$L = \{(x, y) \in S \mid x_0 \leqslant x \leqslant x_1 \text{ and } y \leqslant y_0\}.$$

A priority search tree T for S as defined in [3] is a binary search tree of depth O(log n) for all x-coordinates of points in S. The points themselves are stored in the nodes of T as follows:

(1) The root of T contains the point with the smallest y-coordinate.

(2) If point p is stored in node v ($v \neq$ root), then the search path to the x-coordinate of p goes through v and father(v) contains a point p′ with no larger y-coordinate than p.

Let A denote the set of nodes in T which contain the points of the answer L to the query and let P_0 and P_1 be the search paths to x_0 and x_1 ending in the leaves v_0 and v_1 (see Fig. 1).

For every $v \in A$ there are four possible places in T:

(a) $v \in P_0$,

(b) $v \in P_1$ and $v \notin P_0$,

(c) v is a descendant of a right son of a node on $P_0 - P_1$,

(d) v is a descendant of a left son of a node on $P_1 - P_0$.

Thus, A can be written as the union of four disjoint sets

$$A = A_0 \cup A_1 \cup A'_0 \cup A'_1,$$

where

$$A_0 = \{v \in P_0 \mid x_0 \leqslant v.x \leqslant x_1 \text{ and } v.y \leqslant y_0\}$$

(here, v.x and v.y are respectively the x- and y-coordinate of the point stored in node v),

$$A_1 = \{v \in P_1 \mid u_1.y \leqslant v.y \leqslant y_0 \text{ and } v.x \leqslant x_1\}$$

(here, u_1 is the highest node on P_1 not belonging to P_0, i.e., u_1 is the right son of the lowest common ancestor u of v_0 and v_1; here we assume that all points have distinct y-coordinates; the general case is treated below),

$$A'_0 = \{v \in P_0 \mid u_0.y \leqslant v.y \leqslant y_0, \text{ the right son w of v exists, } w \notin P_0 \text{ and } w.y \leqslant y_0\}$$

(here, u_0 is the highest node on P_0 not belonging to P_1, i.e., u_0 is the left son of the lowest common ancestor u of v_0 and v_1),

A^r_0 = the set of all right descendants v of nodes in A'_0 with $v.y \leqslant y_0$,

$$A'_1 = \{v \in P_1 \mid u_1.y \leqslant v.y \leqslant y_0, \text{ the left son w of v exists, } w \notin P_1 \text{ and } w.y \leqslant y_0\},$$

and

A^ℓ_1 = the set of all left descendants v of nodes in A'_1 with $v.y \leqslant y_0$.

Sets A_0, A_1, A'_0, and A'_1 can each be computed in O(log n) time by simply inspecting all nodes on the search paths P_0 and P_1. Knowing A'_0 and A'_1, sets A^r_0 and A^ℓ_1 can easily be found in $O(|A^r_0| + |A^\ell_1|)$ time, since every node in A'_0 or A'_1 is the root of a subtree containing a compact subset of L. Thus, L can be computed in

$$O(\log n + |A^r_0| + |A^\ell_1|) = O(\log n + |L|)$$

time.

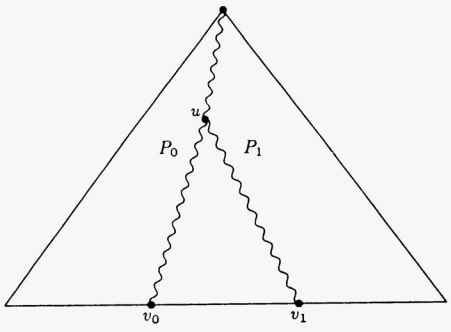

Fig. 1.

3. The extended priority search tree

If the x-coordinates of all points are from the set $\{1, 2, \ldots, N\}$, the leaves v_0 and v_1 corresponding to x_0 and x_1 can be found in constant time and O(N) space by simply using an array that, for every $x \in [1..N]$, contains a pointer to its corresponding leaf. Now we are going to extend the

priority search tree T such that the sets A_0, A_1, A'_0, and A'_1 can be found in

$$O(\log\log n + |A_0| + |A_1| + |A'_0| + |A'_1|)$$

time starting in v_0 and v_1. Thus, we obtain the answer L to the three-sided range query in $O(\log\log n + |L|)$ time.

For each leaf v of T we build four additional priority search trees T_v, T'_v, T^{ℓ}_v, and T^{r}_v. Let

$$P = w_0 — w_1 — w_2 — \cdots — w_k = v$$

be the search path from the root w_0 of T to the leaf v.

(a) T_v is a priority search tree for the set of points stored in the nodes of P.

(b) T'_v is also a priority search tree for the points stored on the path to v but with exchanged coordinates. (T'_v can answer a three-sided range query of the form (y_0, y_1, x_0).)

(c) T^{ℓ}_v is a priority search tree for the set of pairs

$$\{(w_i.y, \mathit{leftsony}(w_i)) \mid 0 \leqslant i \leqslant k\}.$$

(d) T^{r}_v is a priority search tree for the set of pairs

$$\{(w_i.y, \mathit{rightsony}(w_i)) \mid 0 \leqslant i \leqslant k\}.$$

Here

$$\mathit{leftsony}(v) = \begin{cases} w.y & \text{if the left son w of v is not on P,} \\ \infty & \text{otherwise,} \end{cases}$$

and

$$\mathit{rightsony}(v) = \begin{cases} w.y & \text{if the right son w of v is not on P,} \\ \infty & \text{otherwise.} \end{cases}$$

Note that every additional priority search tree is built for a set of O(log n) points and therefore needs O(log n) space and can answer a query in O(log log n) time. The total space requirement is bounded by O(n log n), since we build 4n additional priority search trees of size O(log n) each. In the next section we shall show how the space can be reduced to O(n).

3.1. Lemma. *A_0, A_1, A'_0, and A'_1 can be computed in*

$$O(\log\log n + |A_0| + |A_1| + |A'_0| + |A'_1|)$$

time and $O(N + n \log n)$ *space.*

Proof. We compute A_0, A_1, A'_0, and A'_1 as follows:

Step 1. Find the leaves v_0 and v_1 corresponding to x_0 and x_1 in constant time using an array [1..N] of pointers to the leaves. Let P_0 (P_1) be the path to v_0 (v_1) in T.

Step 2. Compute the lowest common ancestor u of v_0 and v_1 in $O(\log\log n)$ time as described in [1]. Let u_0 (u_1) be the left (right) son of u.

Step 3. Compute

$$A_0 = \{v \in P_0 \mid x_0 \leqslant v.x \leqslant x_1 \text{ and } v.y \leqslant y_0\}$$

by a three-sided range query in priority search tree T_{v_0} in $O(\log\log n + |A_0|)$ time.

Step 4. Compute

$$A_1 = \{v \in P_1 \mid u_1.y \leqslant v.y \leqslant y_0 \text{ and } v.x \leqslant x_1\}$$

by a three-sided range query in priority search tree T'_{v_1} in $O(\log\log n + |A_1|)$ time.

Step 5. Compute

$$A'_0 = \{v \in P_0 \mid u_0.y \leqslant v.y \leqslant y_0 \text{ and } \mathit{rightsony}(v) \leqslant y_0\}$$

by a three-sided range query in $T^{r}_{v_0}$ in $O(\log\log n + |A'_0|)$ time.

Step 6. Compute

$$A'_1 = \{v \in P_1 \mid u_1.y \leqslant v.y \leqslant y_0 \text{ and } \mathit{leftsony}(v) \leqslant y_0\}$$

by a three-sided range query in $T^{\ell}_{v_1}$ in $O(\log\log n + |A'_1|)$ time.

Thus, the total cost for Steps 1 to 6 is

$$O(\log\log n + |A_0| + |A_1| + |A'_0| + |A'_1|). \qquad \square$$

3.2. Remarks. (a) In Steps 4 to 6 we used the y-coordinates of the points to restrict the search on nodes lying below the lowest common ancestor u of v_0 and v_1 on the search paths. This is correct only if all points in S have different y-coordinates.

In the general case, we replace the y-coordinates $w_i.y$, $1 \leqslant i \leqslant k$, of points stored on the search path $w_0, w_1, w_2, \ldots, w_k = v$ by the pairs $(w_i.y, i)$, $1 \leqslant i \leqslant k$, and use the lexicographic ordering on these pairs for searching in T'_v, T^{ℓ}_v, and T^{r}_v. The range bounds for the first dimension in Steps 4 to 6 then change to

$$((u_1.y, j), (y_0, \infty)) \quad \text{and} \quad ((u_0.y, j), (y_0, \infty)),$$

where j is the position of u on the search path to v.

(b) The case where different points may have the same x-coordinate is treated as follows. Let S(x) be the set of all points in S with x-coordinate x. We store S(x) in a heap $H_{S(x)}$. The point of S(x) with minimal y-coordinate is stored in the root of $H_{S(x)}$. Then, build the data structure described above for all heaps $H_{S(x)}$; consider x as the first coordinate and the y-coordinate of the point in the root of $H_{S(x)}$ as the second coordinate. All lists contributing a part to the answer of the query are computed as described in Lemma 3.1 and the answer can be found by simply scanning these heaps until a point is reached whose y-coordinate is greater than y_0.

4. Space reduction

In this section we show how the space requirement of the extended priority search tree can be reduced from O(n log n) to O(n).

4.1. Theorem. *Let* S *be a set of* n *points in* $[1..N] \times \mathbb{R}$. *A three-sided range query* (x_0, x_1, y_0) *on* S *can be answered in* $O(\log \log n + k)$ *time,* $O(N + n)$ *space and* $O(N + n)$ *preprocessing time* (k *is the size of the answer*).

Proof. Lemma 3.1 gives a solution with $O(N + n \log n)$ space. We now show how to reduce the space of the extended priority search tree to O(n). Let T be a priority search tree for S and B be the set of all nodes on level log log n (counted from the bottom) (see Fig. 2). Clearly, $|B| = O(n/\log n)$ and each node $v \in B$ is the root of a subtree with O(log n) leaves and height O(log log n). Now, we construct the four additional priority search trees for every node $v \in B$. Since every path from the root to v has length $O(\log n - \log \log n)$ and there are $O(n/\log n)$ nodes in B, these additional priority search trees need only O(n) space. Thus, the total space requirement is O(n).

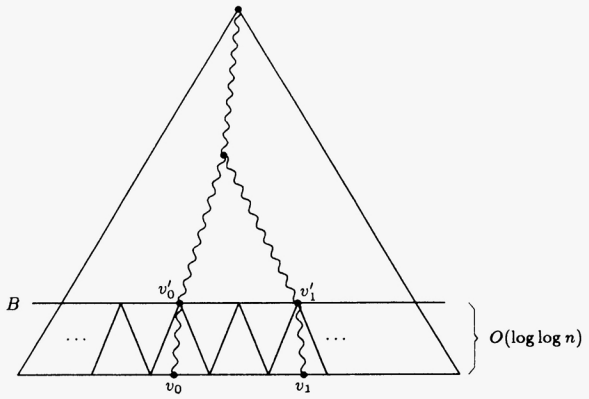

Fig. 2.

The cost for building the entire data structure is $O(N + n)$. We first sort our points according to x-coordinate in $O(N + n)$ time using bucket sort. This yields the set S(x) for every possible x-coordinate and also, in the same time bound, the pair with minimal y-coordinate in S(x). At this point we arrive at a set (x_i, y_i), $1 \leqslant i \leqslant n' \leqslant n$ and $x_i < x_{i+1}$ for all i. We may also assume that the y-coordinates are pairwise distinct by Remark 3.2(a).

Next, we shall show that we can build a priority search tree for a set of points (a_i, b_i), $1 \leqslant i \leqslant n$ and $a_i < a_{i+1}$ for all i, in O(n) time. To build a search tree with n leaves we need O(n) time. In the next step, we store the points in the leaves corresponding to their x-coordinate. Next, consider a node v such that the two sons of v already contain a point. We move the point with the smallest y-coordinate stored in one of the sons of v to v. Now, we go to this son and repeat the procedure in exactly the same way. Thus, we can fill a node v in time proportional to the height of v. Since the number of nodes of height h is $n/2^h$ we need O(n) time to construct the priority search tree.

For our data structure we need five priority search trees: the primary tree and the four auxiliary trees T_v, T'_v, T^{ℓ}_v, and T^{r}_v for every vertex v of height log log n. From the previous paragraph, we can clearly construct the primary tree in O(n)

time. We can also build all the heaps $H_{S(x)}$ in O(n) time and, finally, we can provide the pointers from the array representing the universe to the leaves of the primary tree in O(N) time.

Let us consider the auxiliary trees next. The trees T'_v, T'^{ℓ}_v, and T'^{r}_v use the y-coordinate as the major coordinate. Since they are constructed for the pairs on a path in T, the input is sorted by y-coordinate and hence we can construct each tree in O(log n) time or O(n) total time. Recall that there are O(n/log n) different vertices $v \in B$.

The auxiliary tree T_v uses the x-coordinate as the major coordinate. Number the vertices $v \in B$ from left to right in increasing order. For every vertex $v \in B$ and vertex w on the path from the root of T to v create the pair (*number*(v), w.x). Sort these pairs into lexicographic order in O(N + n) time using bucket sort. This gives us, for every $v \in B$, the points to be stored in T_v sorted according to x-coordinate. Thus, the trees T_v, $v \in B$, can also be constructed in O(n) total time.

A query is now performed as follows:

Step 1. Find the leaves v_0 and v_1 using the array in constant time.

Step 2. Find v'_0, $v'_1 \in B$ such that v'_0 is an ancestor of v_0 and v'_1 is an ancestor of v_1 in O(log log n) time (just climb up the tree starting in v_0 and v_1 until B is reached).

Step 3. Compute the answer as described in Lemma 3.1 with v_0, v_1 replaced by v'_0, v'_1.

Step 4. It remains to compute the part of the answer lying in the subtrees rooted at v'_0 and v'_1. But, these subtrees can be regarded as two conventional priority search trees of O(log log n) height and their part L′ to the answer can be computed in O(log log n + |L′|) time by the standard method described in Section 2. □

References

[1] D. Harel and R.E. Tarjan, Fast algorithms for finding nearest common ancestors, SIAM J. Comput. 13 (1984) 338–355.

[2] K. Mehlhorn, Data Structures and Algorithms, Vol. 3: Multidimensional Searching and Computational Geometry (Springer, Berlin, 1984).

[3] E.M. McCreight, Priority search trees, Tech. Rept. Xerox PARC CSL-81-5, 1981.

[4] M.H. Overmars, Range searching on a grid, Proc. WG'85, Internat. Workshop on Graph-Theoretic Concepts in Computer Science, 1985.

[5] F.P. Preparata and M.I. Shamos, Computational Geometry, An Introduction (Springer, Berlin, 1985).