A DATA STRUCTURE FOR DYNAMIC RANGE QUERIES

by

George S. Lueker[+]

Department of Information and Computer Science
University of California, Irvine
Irvine, CA 92717

Technical Report #123
September, 1978

This is a revised version of the paper "A Data Structure for Orthogonal Range Queries" which is to appear in the Proceedings of the Nineteenth Annual IEEE Foundations of Computer Science Conference, October 16-18, 1978.

Key words and phrases:

Range queries
File organization
Trees of bounded balance
Data base management
Data structures
Searching

CR categories:

3.73,3.74,4.33,
4.34,5.25

[+]Partially supported by NSF Grant MCS77-04410.

A Data Structure for Dynamic Range Queries

Abstract

Given a set of points in a k-dimensional space, an orthogonal range query is a request for the number of points in a specified k-dimensional box. We present a dynamic data structure and algorithm which enable one to insert and delete points and to perform orthogonal range queries. The worst-case time complexity for n operations is $O(n \log^k n)$; the space used is $O(n \log^{k-1} n)$. (O-notation here is with respect to n; the constant is allowed to depend on k.) This is faster than the best previous algorithm by a factor of log n; the data structure also handles deletions in a more general context than previous fast algorithms.

1. Introduction

A number of highly efficient data structures have been devised which make it possible to manipulate a set of records with totally ordered keys. For example, suppose one wishes to be able to insert, delete, or locate a given record in a set of n elements. These operations can all be performed in an average of O(log n) time per operation through the use of binary search trees [K73, AHU74]; by using 2-3 trees, AVL trees, or trees of bounded balance, the time bound can be improved to worst-case O(log n) [AL62, K73, NR73, AHU74]. These trees also allow more general operations to be performed [see C72, K73, AHU74]. For example, we may wish to perform a "range query," that is, to determine the number of keys in the range [a,b]; it is not hard to devise a worst-case O(log n) algorithm to do this using any of the three balanced schemes mentioned above, provided each node has a field containing its number of

descendants.

Now suppose each key is a vector of length k; more complex queries become possible. For example, we may wish to specify a range for each component of the key and ask how many keys have all components in the desired range. This corresponds to counting the number of elements in some k-dimensional box; Knuth [K73, pp. 554-555] calls this an orthogonal range query; unfortunately, he observes that for this problem "No really nice data structures seem to be available." (An interesting special case of range queries is partial match queries; these have been investigated by a number of researchers, with considerable success. See [R76].)

Since then, a substantial amount of progress has been made. In order to discuss it, some terminology will be useful. Suppose we have n points in k-space and wish to perform operations, such as insertions and queries, on them. Let S(n) denote the total amount of space used; let Q(n) be the time required to respond to a single query; finally let P(n) denote the total amount of time spent processing the insertions (and deletions, if any). If all of the points must be presented before any queries are presented, so that the set of points is not allowed to change, we say the data structure is static; in this case it does not even make sense to discuss deletions. If queries may be interspersed with insertions (and deletions, if allowed) we say that the data structure is dynamic. In the dynamic case it is more

convenient to let n be the number of operations we perform on the data structure. Note that the number of elements present at any time is bounded by n. Finally, it is useful to discuss a generalization of the problem discussed above. Suppose that instead of merely wishing to know the number of records in a given range, we wish to know some function of that set of records. Suppose that in addition to a k-dimensional key, each record in our data structure has a field VALUE, and that we wish to find the sum of the VALUE fields of all records with keys in the specified range. More generally, suppose that we wish to find the result of combining the VALUE fields under some operator $\square$. For example, to find the minimum salary of people in some age range, we could let $\square$ be the operator min. If $\square$ is commutative and associative, can be computed in O(1) time, and works on objects which can be represented in O(1) space, we will call it an admissible operator. Note for example that the operators "+" and "min" are admissible.

One data structure which has been proposed for handling range queries is the quad tree [FB74]. A k-dimensional quad tree is much like a binary search tree, except that each node has up to 2^k children; thus when a new node is inserted, its relation to its parent can depend on the outcome of comparisons of all k components of the keys. A closely related data structure called a multidimensional binary search tree has been introduced [B75]; this data structure also allows one to perform orthogonal range queries. It has been shown that for either data structure,

if the trees involved are balanced, an orthogonal range query can be performed in a worst case time of $O(n^{1-1/k})$ [BS75, LW77]; analysis of the average performance appears to be more difficult.

Recently significant advances have been made in the development of orthogonal range query algorithms and data structures. (For all of the data structures in this paragraph, we are allowing insertions and queries, but not deletions. However, it has been observed [B78b] that if $\square$ is an admissible operator which admits inverses, the following simple scheme enables one to simulate deletions. Maintain two data structures, one containing all records which have been inserted, and one containing all records which have been deleted. Respond to a query by returning the "difference" under $\square$ of the responses to the two data structures.) In [BS77] a static structure is presented which makes it possible to perform range queries, assuming the desired response is simply a count of the number of records in the specified range; the performance measures are

$$P(n) = S(n) = O(n \log^{k-1} n)$$

$$Q(n) = O(\log^{k} n)$$

A key idea used in that paper is multidimensional divide-and-conquer; see [B78a,BS76] for a thorough discussion of several applications of this fundamental technique. In [B77] a very elegant approach to a variety of searching problems is developed. The notion of a

decomposable searching problem is defined, which encompasses a large number of problems. (Range queries with admissible operators are a special case of the notion of decomposable searching problems.) Techniques are developed by which one-dimensional range restrictions can be added to any data structure for a decomposable problem, while increasing the space and time complexity by at worst a factor of O(log n). This enables one to immediately obtain the result of [BS77] quoted above, but for any admissible operator. Also presented is a technique for converting a static structure into a dynamic structure without increasing the storage space, and with only a factor of O(log n) increase in the time complexity. This result, combined with the bounds previously mentioned, enables one to produce a dynamic data structure for range queries with

$$\left.\begin{aligned} P(n) &= O(n \log^{k} n) \\ S(n) &= O(n \log^{k-1} n) \\ Q(n) &= O(\log^{k+1} n) \end{aligned}\right\} \qquad (1)$$

These are the previous best known bounds for arbitrary dynamic range query problems. However, if it is known in advance that either insertions will be much more frequent than queries or vice versa, it is possible to obtain faster overall times by use of the results in [BM78], where static data structures are presented for which

$$P(n) = S(n) = O(n^{1+\epsilon})$$

$$Q(n) = O(\log n)$$

or, alternatively,

$$P(n) = O(n \log n)$$

$$S(n) = O(n)$$

$$Q(n) = O(n^{\epsilon})$$

for any $\epsilon > 0$. See [BF78] for a survey of known algorithms for range searching.

In this paper we make three contributions:

a) We show how to improve the time bound for dynamic range searching to $Q(n) = O(\log^k n)$ without increasing S or P over the bounds in (1).

b) We develop a data structure which allows deletions to be performed efficiently under any admissible operator.

c) We show how the notion of trees of bounded balance [NR73] is relevant to the problem of range searching.

In section 2 we review the results of [B77,BS77] in somewhat greater detail and lay the foundations for our approach. In section 3 we show how to make the structure dynamic efficiently in a way which allows deletions.

2. Range trees and k-fold binary search trees

The following definition gives a convenient way of talking about the results of [B77, BS77].

Define a k-fold binary search tree, on a set of n k-dimensional keys, inductively as follows. If k=1, a k-fold binary search tree is an ordinary binary search tree; since k=1, keys have only one component, and the tree is ordered according to this component. In addition, when k=1, each node has a field SUM which tells the "sum" under $\square$ of the VALUE field of all of the records which descend from it. If k>1, then a k-fold binary search tree on n records is a binary search tree organized according to the k^{th} components of keys; in addition, however, each node x contains a field AUX which points to a (k-1)-fold binary search tree containing all records which descend from x, organized according to the first k-1 components of the keys. The n nodes in the binary tree organized according to the k^{th} component are called primary nodes; the nodes in all of the other trees are called secondary nodes. If x is a primary node in a k-fold binary search tree, we say x has dimension k, or more briefly, dim(x)=k. An algorithm for inserting in a k-fold binary search tree is shown below. (The set of descendants of x is considered to include x; it does not include any of the nodes in AUX(x). Similar remarks apply to the term ancestor.)

```
procedure INSERT(k,K,T);
begin
  comment insert a key K into a k-fold binary search tree
    T;
  create a new node x for key K and insert it into T
    according to component k, thinking of T as a binary
    search tree;
  for each ancestor y of x do
    if k = 1
      then SUM(y) := SUM(y) □ VALUE(x)
      else INSERT(k-1,K,AUX(y));
end;
```

Note that k-fold binary search trees may be used as the basis of an algorithm to respond to an orthogonal range query. First, find the set of subtrees in T which corresponds to records whose k^{th} component satisfies the query; let R be the set of roots of these trees. If T is well balanced, O(log n) is a bound on the size of R and the time to compute R. Now sum the results of orthogonal range queries on the remaining k-1 dimensions for the auxiliary trees of all of the nodes in R. If T is well balanced, an easy induction shows that this algorithm runs in $O(\log^k n)$ time. If for each node in the tree, the number of nodes in the left and right subtrees differs by at most one, the tree is essentially the range tree discussed in [B77, BS77]; it is shown there that range trees can be constructed in $O(n \log^{k-1} n)$ time; this is the method used there to achieve a fast range query data structure.

3. Algorithms with partially balanced trees

In this paper, we will not require that the trees be balanced as strictly as in [B77,BS77]. We begin by showing that if keys are inserted at random according to algorithm INSERT

above, T will be fairly well balanced.

Theorem 1. Assume the components of keys are chosen independently and randomly, in such a way that all keys are distinct in each component, and in each component the permutation required to sort the keys is equally likely. Then the expected time to insert n keys into a k-fold binary search tree is $O(n \log^k n)$.

Proof. We use induction on k. For k=1, clearly $T(n,k)=O(n \log n)$. Now let k>1. Since the expected internal path length of T is $O(n \log n)$, the expectation of the total number of records inserted in auxiliary trees of primary nodes of T is $O(n \log n)$. By the induction hypothesis, the time to insert m keys in one of these auxiliary trees is $O(m \log^{k-1} m)$, which is $O(m \log^{k-1} n)$. Thus the time for all of the $O(n \log n)$ insertions must be $O(n \log^k n)$. []

Now we will introduce a data structure which makes it possible to perform a sequence of n insertions, deletions, and orthogonal range queries in worst-case $O(n \log^k n)$ total time. The basic idea is to use a balanced tree scheme. A problem that arises is that rebalancing a node may require the associated auxiliary tree to be completely restructured; this can be expensive for large trees. We would like to guarantee that this will happen only very rarely for trees with many nodes. Below we will show how trees of bounded balance [NR73] can be used to produce the desired data structure.

Let the rank of a node x, written rank(x), be one more than the number of nodes which descend from x. The balance of a node x, written $\rho(x)$, is the ratio of the rank of the left child of x to rank(x). A node x is α-balanced if

$$\rho(x) \in [\alpha, 1-\alpha].$$

In [NR73] it is shown that if a tree on n nodes is α-balanced for some positive α, then the height of the tree is O(log n). It is also shown that, assuming $\alpha < 1 - \sqrt{2}/2$, balance in a tree may be maintained through the use of some simple rebalancing operations, so that insertions and deletions can be done in O(log n) worst-case time per operation. (We henceforth assume $\alpha \leq 1-\sqrt{2}/2$.)

In order to cast the results of [BS77] in terms of bounded balance, we say a k-fold binary search tree is a k-fold BB(α) tree if the balance of each primary or secondary node is in $[\alpha, 1-\alpha]$. It is easy to see that for such a tree,

$$Q(n) = O(\log^k n)$$

$$S(n) = O(n \log^{k-1} n).$$

Next we discuss the cost of insertions and deletions.

Lemma 1. Insertion or deletion of a node in a binary search tree on m records can change the balance of the root by only O(1/m).

The proof is easy and is omitted.

The algorithm INSERTB for insertion in a k-fold BB(α) tree is given below; deletion can be handled similarly. (In this algorithm we have not explicitly shown the updating of the SUM fields; this can be done in a manner similar to that shown in the INSERT procedure above.)

```
procedure INSERTB(k,K,T);
begin
  comment insert key K into a k-fold BB(α) tree T,
    rebalancing as needed to maintain the BB(α) condition;
  if k=1
    then insert K in T using the standard bounded balance
      insertion procedure
    else
      begin
        insert a new node x for key K into T according to
          component k of K, thinking of T as a binary
          search tree;
        for each ancestor y of x do
          INSERTB(k-1,K,AUX(y));
        if some node on the path from the root of T to x
          has balance outside of [α,1-α] then
            REBALANCE: begin
              locate the first node y on the path from the
                root of T to x for which ρ(y) ∉ [α,1-α];
              replace the subtree rooted at y by a range
                tree for all of the records stored in that
                subtree;
            end;
      end;
end;
```

Theorem 3. In the worst case, INSERTB uses $O(n \log^k n)$ time for n insertions.

Proof. Let T(n,k) be the worst-case time required to perform n insertions on a k-fold BB(α) tree. It is clear that

$$T(n,1) = O(n \log n).$$

Now suppose k>1. Aside from the time spent in the block labeled REBALANCE, the time bound is easy. We will use an accounting argument to establish the time bound for the REBALANCE block. We begin with a few definitions. For any primary or secondary node x, let $\beta(x)$ be the distance on the real line from the point $\rho(x)$ to the set [1/3,2/3]; that is,

$$\beta(x) = \max(0, \rho(x)-2/3, 1/3-\rho(x))$$

Let dim(x) be the dimension of the subtree of which x is a root. Now we define the imbalance I(T) of a tree T to be the sum, over all primary and secondary nodes x in T with dim(x)>1, of

$$\beta(x)\ \mathrm{rank}(x)\ \log^{\dim(x)-1} n$$

We will charge each insertion operation an amount equal to the increase it produces in the imbalance of T, before any rebalancing is performed. In this paragraph we prove by induction on k that a single insertion can increase I(T) only by $O(\log^k n)$. First note that by Lemma 1, $\beta(x)$ rank(x) can be seen to increase by at most O(1) per insertion for any node in the tree. Thus for k=2, the increase in I(T) during an insertion is $O(\log^2 n)$, since only nodes on the path from the root to the inserted node are affected. Now we prove the assertion for dimension k, assuming it holds for lower dimensions. The increase of $\beta(x)$ rank(x) for a primary node in T is O(1) as in the basis, so the change in I(T) due to these nodes is $O(\log n \cdot \log^{k-1} n)$ or $O(\log^k n)$.

However, we also insert the new key into O(log n) auxiliary trees, each of dimension k-1. By the induction hypothesis, the increase in I(T) for each such insertion is $O(\log^{k-1} n)$, for a total of $O(\log^{k} n)$. Thus the total increase in I(T) for a k-fold BB(α) tree due to a single insertion is $O(\log^{k} n)$.

Next we show that the decrease in I(T) due to a REBALANCE operation at node y is sufficient to cover the cost of the operation. Note that after the operation, β is $\emptyset$ for all primary or secondary nodes in the new subtree, since in a range tree each node has ρ in [1/3,2/3]. On the other hand, we are performing the operation since node y had a balance outside $[\alpha,1-\alpha]$. Thus, initially, we had

$$\beta(y) > \alpha - 1/3.$$

Let r = rank(y) and d = dim(y). Then the decrease in I(T) is at least

$$(\alpha-1/3)\ r\ \log^{d-1} n.$$

On the other hand, from [B77, BS77] we know that the time required to construct the range tree is

$$O(r\ \log^{d-1} r).$$

Thus by choice of suitable constants in the accounting argument, we can guarantee that the decrease in imbalance covers the cost.

Since the imbalance of an initial empty tree is $\emptyset$, and never goes negative, we have shown that the amount charged to the operations covers the rebalancing costs. This completes the proof. []

Now we briefly explain how deletions can be performed. We use the method described for deletions from a binary search tree described in [AHU74], followed by updates on the AUX and SUM fields, and rebalancing. The AUX fields are updated in much the same way as in procedure INSERTB; when a node is inserted (respectively deleted) at some point in the tree, a corresponding insertion (respectively deletion) must be made in the AUX fields of all of its ancestors. It is tempting to say that deletion of a node x which descends from a node y, with dim(y)=1, should be followed by the operation

$$SUM(y) := SUM(y) \square VALUE(x)^{-1}$$

This is disallowed, however, since we have no guarantee that $\square$ admits inverses. Instead, we calculate SUM(y) by combining the SUM fields of the children of y under $\square$. An argument much like that used for insertions can now be used to show that the time bound still holds.

4. Conclusions

We have presented an algorithm for dynamic range query problems which is faster than the previous best known approach by a factor of log n; it also allows deletions to

be handled readily. Unfortunately, our approach is somewhat more ad hoc than that used in [B77]; we have not developed a general transformation which can be used to add range restriction capability to an arbitrary dynamic data structure for a decomposable problem. We plan to address this topic in another paper.

It should be noted that although we used trees of bounded balance, the rebalancing operations discussed in [NR73] are not essential to our data structure. When rebalancing a node of dimension greater than 1, we completely restructure the tree; further, for trees of dimension one, we could use any balanced tree scheme which guarantees logarithmic behavior.

References

[AL62] G. M. Adel'son-Vel'skiĭ and E. M. Landis, "An Algorithm for the Organization of Information," Sov. Math. Dokl., 3 (1962), pp. 1259-1262.

[AHU74] Alfred Aho, John Hopcroft, and Jeffrey Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, Mass., 1974.

[B75] Jon L. Bentley, "Multidimensional Binary Search Trees Used for Associative Searching," CACM, 18:9 (1975), pp. 509-517.

[B77] Jon L. Bentley, "Decomposable Searching Problems," unpublished technical report, Carnegie-Mellon University, 1977.

[B78a] Jon L. Bentley, "Multidimensional Divide-and-Conquer," Carnegie-Mellon University Computer Science Research Review, to appear.

[B78b] Jon L. Bentley, private communication, August 1978.

[BF78] Jon L. Bentley and Jerome H. Friedman, "A Survey of Algorithms and Data Structures for Range Searching," Proc. of the Computer Science and Statistics: Eleventh Annual Symposium on the Interface, (March 1978), pp. 297-307.

[BM78] Jon L. Bentley and H. A. Maurer, "Efficient Worst-Case Data Structures for Range Searching," draft.

[BS76] Jon L. Bentley and Michael I. Shamos, "Divide-and-Conquer in Multidimensional Space," Proc. 8th Annual ACM Symposium on Theory of Computing, (May 1976), pp. 220-230.

[BS77] Jon L. Bentley and Michael I. Shamos, "A Problem in Multivariate Statistics: Algorithm, Data Structure, and Applications," Proceedings of the Fifteenth Allerton Conference on Communication, Control, and Computing, (September 1977), pp. 193-201.

[BS75] Jon L. Bentley and Donald F. Stanat, "Analysis of Range Searches in Quad Trees," Inf. Proc. Letters, 3:6 (1975), pp. 170-173.

[C72] C. A. Crane, "Linear Lists and Priority Queues as Balanced Binary Trees," Ph. D. Thesis, Stanford University, 1972.

[FB74] R. A. Finkel and J. L. Bentley, "Quad Trees: A Data Structure for Retrieval on Composite Keys," Acta Inf., 4 (1974), pp. 1-9.

[K73] Donald Knuth, The Art of Computer Programming, Vol. 3: Sorting and Searching, Addison-Wesley, Reading, Mass., 1973.

[LW77] D. T. Lee and C. K. Wong, "Worst-Case Analysis for Region and Partial Region Searches in Multidimensional Binary Search Trees and Balanced Quad Trees," Acta Inf., 9 (1977), pp. 23-29.

[NR73] J. Nievergelt and E. M. Reingold, "Binary Search Trees of Bounded Balance," SIAM J. Comput., 2:1 (1973), pp. 33-43.

[R76] Ronald Rivest, "Partial Match Retrieval Algorithms," SIAM J. Comput., 5:1 (1976), pp. 19-50.