M. Jeffery, G. Iba, M. Hornell, and K. Prendergast. Later versions were improved by discussions with R. Berwick, M. Brady, and other participants in the Artificial Intelligence Laboratory's Learning Seminar. The drawings are by K. Prendergast.

Received 8/79; revised 6/80; accepted 7/80.

References

1. Brotsky, D. Efficient graph matching through exploitation of constraint. M.I.T. Artif. Intell. Lab. Memo No. 600, Cambridge, Mass., Oct. 1980.
2. Carroll, J.B., Daves, P., and Richmond, B. *Word Frequency Book.* Houghton-Mifflin and American Heritage, New York, 1971.
3. Evans, T.G. A heuristic program to solve geometric analogy problems. Ph.D. Th., M.I.T., Cambridge, Mass. in *Semantic Information Processing*, M. Minsky, Ed., The M.I.T. Press, Cambridge, Mass., 1968.
4. Filmore, C.J. The case for case. In *Universals in Linguistic Theory*, E. Bach and R. Harms, Eds., Holt, Rinehart, and Winston, New York, 1968.
5. Givon, T. Cause and control: On the semantics of interpersonal manipulation. In *Syntax and Semantics*, Vol IV, J. Kimball, Ed., Academic, New York, 1975.
6. Katz, B. A three-step procedure for language generation. M.I.T. Artif. Intell. Lab. Memo. No. 599, Cambridge, Mass. Oct. 1980.
7. Lenat, D. AM: An artificial intelligence approach to discovery in mathematics as heuristic search. Ph.D. Th., Stanford Univ., Stanford, Calif. in *Knowledge-Based Systems in Artificial Intelligence*, McGraw-Hill, New York, 1979.
8. Martin, W.A. Philosophical foundations for a linguistically oriented semantic network (in preparation).
9. Meldman, J. A preliminary study in computer-aided legal analysis. Ph.D. Th. and Tech. Rep. No. MAC-TR-157, M.I.T. Lab. for Comptr. Sci., Cambridge, Mass., Nov. 1975.
10. Minsky, M. A framework for representing knowledge. In *The Psychology of Computer Vision*, P.H. Winston, Ed., McGraw-Hill, New York, 1975.
11. Moore, J. and Newell, A. How can Merlin understand? In *Knowledge and Cognition.* L. Gregg, Ed., Lawrence Erlbaum Associates, Potomac, Md., 1974.
12. Ogden, C.K. *Basic English: International Second Language.* Harcourt, Brace, and World, New York, 1968.
13. Rieger, C. The commonsense algorithm as a basis for computer models of human memory, inference, belief, and contextual language comprehension. Dept. Comptr. Sci. Tech. Rep. No. 373, Univ. of Maryland, College Park, Md., 1975.
14. Roberts, R.B. and Goldstein, I.P. The FRL primer. M.I.T. Artif. Intell. Lab. Memo No. 408, Cambridge, Mass., July 1977.
15. Roberts, R.B. and Goldstein, I.P. The FRL manual. M.I.T. Artif. Intell. Lab. Memo No. 409, Cambridge, Mass., June 1977.
16. Rosch, E., Mervis, C.B., Gray, W.D., Johnson, D.M., and Boyes-Braem, P. Basic objects in natural categories. *Cog. Psych 8,* 3 (July 1976) 382–439.
17. Schank, R.C. *Conceptual Information Processing.* North-Holland, New York, 1975.
18. Tversky, A. Features of similarity. *Psych. Rev. 84,* 4 (July 1977), 327–352.
19. Wilks, Y.A. *Grammar, Meaning, and the Machine Analysis of Language.* Routlege and Kegan Paul, London, 1972.
20. Winston, P.H. Learning structural descriptions from examples. Ph.D. Th., M.I.T., Cambridge, Mass. in *The Psychology of Computer Vision*, P.H. Winston, Ed., McGraw-Hill, New York, 1975.
21. Winston, P.H. Learning by creating and justifying transfer frames. *Artif. Intell. 10,* 2 (1978), 147–172.
22. Winston, P.H. Learning and reasoning by analogy: The details (formerly titled "Learning by understanding analogies"). M.I.T. Artif. Intell. Lab. Memo No. 520, Cambridge, Mass., April 1979.

Programming Techniques and Data Structures

R. Rivest Editor

Deletion in Two-Dimensional Quad Trees

Hanan Samet
University of Maryland

An algorithm for deletion in two-dimensional quad trees that handles the problem in a manner analogous to deletion in binary search trees is presented. The algorithm is compared with a proposed method for deletion which reinserts all of the nodes in the subtrees of the deleted node. The objective of the new algorithm is to reduce the number of nodes that need to be reinserted. Analysis for complete quad trees shows that the number of nodes requiring reinsertion is reduced to as low as $\frac{2}{3}$ of that required by the old algorithm. Simulation tests verify this result. Reduction of the number of insertions has a similar effect on the number of comparison operations. In addition, the total path length (and balance) of the resulting tree is observed to remain constant or increase slightly when the new algorithm for deletion is used, whereas use of the old algorithm results in a significant increase in the total path length for large trees.

Key Words and Phrases: binary tree deletion, quad trees, associative searching, information retrieval, binary search trees, sorting, searching, geographic databases

CR Categories: 3.70, 4.34, 4.79

Introduction

A number of data structures have been proposed for retrieval on composite keys [1, 3, 8]. We are interested in the quad tree of [3]. It is useful whenever there is a need to perform operations such as searching a two-dimensional structure. For example, we may wish to find all

Author's present address: H. Samet, Computer Science Department, University of Maryland, College Park, MD 20742.

This research was supported in part by a General Research Board Faculty Award of the University of Maryland and by the Defense Advanced Research Projects Agency and the U.S. Army Night Vision Laboratory under contract DAAG-76C-0138 (DARPA Order 3206).

Fig. 1. A quad tree and the records it represents.

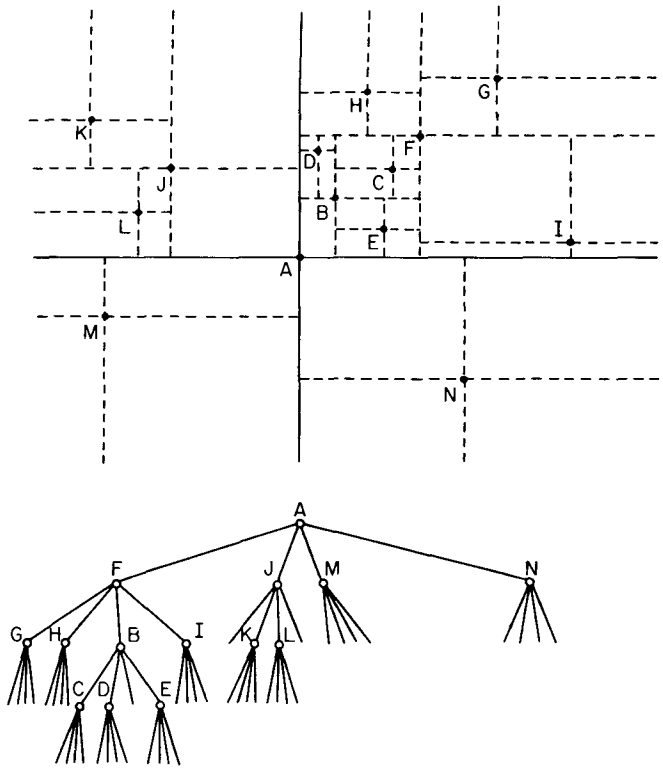
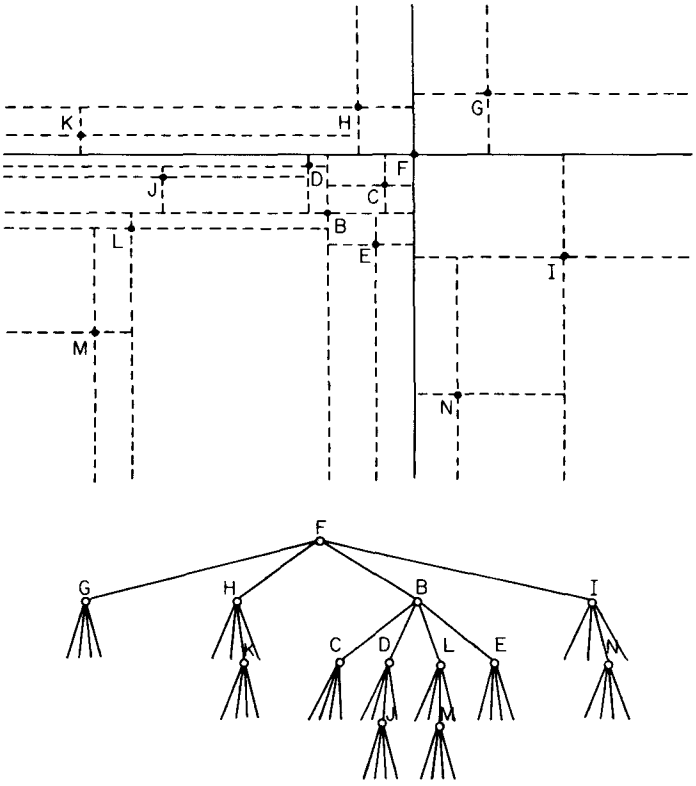


Fig. 2. Result of deleting node A from the quad tree of Figure 1 using the method of [3].



nodes within a specified distance of a given node—i.e., all cities within 50 miles of Washington, D.C. The quad tree is essentially a generalization of a binary tree where each node has four subtrees labeled 1, 2, 3, and 4 corresponding to the directions NE, NW, SW, and SE, respectively. Each subtree is commonly referred to as a quadrant or subquadrant. For example, see Figure 1 where the correspondence of a quad tree of 14 nodes to the records it represents is presented.

In [3] algorithms for insertion, point search, and region search are given. Deletion is resolved by indicating that all of the nodes of the tree rooted at the deleted node must be reinserted—a sometimes expensive process. For example, see Figure 2 where the quad tree which results after deletion of A from the quad tree in Figure 1 is shown. In [3] it is suggested that one should not reinsert subtrees elementwise if their new position is known for the subtree as a whole. In this paper we present an improved algorithm for deletion in two-dimensional quad trees which takes advantage of this idea. Section 2 contains the algorithm and an intuitive discussion of its motivation. In Section 3 we analyze the algorithm and in Section 4 we discuss results of simulations. Conclusions are drawn in Section 5.

2. Algorithm

We view deletion in quad trees in a manner analogous to deletion in binary search trees [4, 5]. For example, in the binary search tree of Figure 3 when A is to be deleted, it is replaced by one of B or C—i.e., the "closest" node in value. In the case of a quad tree, as shown in Figure 1, it is not clear which of the remaining nodes should replace A. This is because no definition of "closest" exists. We notice that no matter which of the nodes is chosen, some of the nodes will have to shift position. Therefore, in Figure 1, if L replaced A, then J would no longer occupy the second quadrant with respect to L and thus, J would need to be reinserted along with its subtrees in the quadrant rooted at F. Similarly, E and I would have to be reinserted along with their subtrees in the quadrant rooted at N.

The problem of finding a replacement node is further compounded when the replacing node is not a terminal node. This situation does not arise in binary search trees, but in quad trees it may arise depending on the measure of "closeness" employed. For example, in Figure 1 if the measure of "closeness" is a minimum sum of horizontal

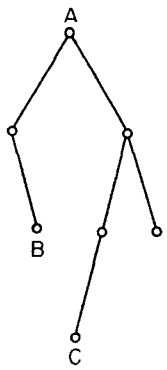


Fig. 3. A binary search tree.

Fig. 4. A quad tree in which deletion requires no reinsertion.

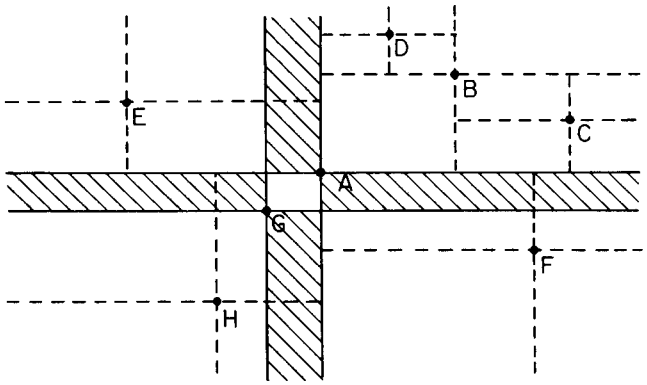
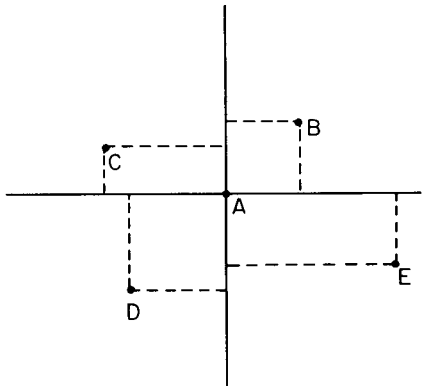


Fig. 5. A quad tree with no "closest" terminal node.



and vertical displacements (i.e., the $L1$ metric) from the node to be deleted, then B is closest to A, yet it is not a terminal node. However, B, does not have a subtree in the quadrant whose boundary it shares with A. Replacing A by B means that subtrees rooted at A must be reinserted in the new tree. This problem can be eased by exploiting certain geometrical properties. For example, in Figure 1, replacing A by B means the the subtree rooted in the third quadrant of the tree rooted at A need not be examined for reinsertion—i.e., it can become the third quadrant of the new tree rooted at B.

Ideally, we wish to replace the deleted node with a node which is terminal and closer to its bordering axes than any other node in the tree. Such a node can simply replace the deleted node with no need for any reinsertion. For example, see Figure 4 where G has no subtrees and is closer than any other node to its bordering sides of the x and y axes. Thus, when G becomes the root of the tree, the crosshatched region in Figure 4 is empty and no nodes need to be reinserted. This is analogous to the binary search tree since the empty crosshatched region corresponds to a gap between the deleted node and its replacement. We use the term "crosshatched" to denote the area between the axes formed by the deleted and replacing nodes. Two items are worthy of further note. First, the condition of the replacing node (e.g., G in Figure 4) being the closest terminal node is not sufficient to guarantee the emptiness of the crosshatched region (it only guarantees the emptiness of the region in the subquadrant rooted at the replacing node). Second, it is not uncommon that such a node fails to exist (see Figure 5). This corresponds to the situation that for any node, say B, which is closest to one of its bordering axes, there exists another node, say C, which is closer to the other axis.

In general, we do not want to expend the time necessary to find the node, if any, which partitions the space so that the crosshatched region is empty. Any algorithm we develop must rely on a method which finds a node satisfying some predetermined criterion of "closeness." Our algorithm relies on a criterion analogous to the one used in a binary search tree to determine a "closest" node.

Let us define the notion of a direction opposite a given direction—the conjugate, as in [3], being

$$\text{conjugate}(N) = ((N + 1) \bmod 4) + 1$$

where N is a quadrant number in the range 1 to 4.

For each quadrant i, of the tree rooted at the node to be deleted, we perform the following procedure:

FINDCANDIDATE: Starting at the root node of the quadrant, repeatedly follow the branch corresponding to conjugate (i) until a node having no subtree along this branch is encountered.

The above procedure results in a set of four nodes which are said to be candidates for replacing the deleted node. Once such a set is found (if no candidate is found in a quadrant, then we choose a fictitious point in the quadrant that is farthest from the deleted node—e.g., $(-\infty, -\infty)$ in the third quadrant), choose the candidate that is closer to each of its bordering axes than any other candidate which is on the same side of these axes. This is termed property (1). For example, in Figure 1 the candidates are B, J, M, N, with B being the winner. The situation that none of the candidates satisfy this criterion (e.g., Figure 5) or that several candidates satisfy this condition (e.g., B and D in Figure 6) may arise. In this case we choose the candidate with the minimum $L1$ metric value (hereafter characterized as property (2)).

As a justification for the use of the $L1$ metric, assume that the nodes are uniformly distributed in the two-dimensional space. The goal is to minimize the area of the crosshatched region. For example (see Figure 7), if the two-dimensional space is finite having sides of length Lx and Ly along the x and y axes, respectively, and a

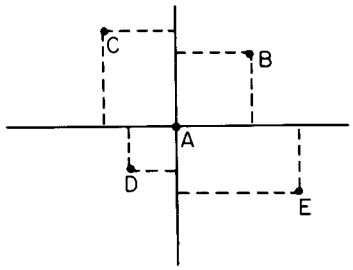
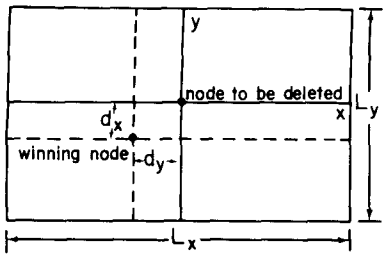


Fig. 6. A quad tree with two nodes being closest to their bordering axes.

Fig. 7. Example of a two-dimensional space.



candidate is at a distance of dx and dy from the x and y axes, respectively, centered at the node to be deleted, then the crosshatched region has an area of $Lx \cdot dx + Ly \cdot dy - 2 \cdot dx \cdot dy$. However, as Lx and Ly increase (as occurs in arbitrary space), the contribution of the $2 \cdot dx \cdot dy$ term becomes negligible and the area is proportional to the sum of dx and dy.

Property (2) is not sufficient by itself to insure that the selected candidate partitions the space so the crosshatched region contains no other candidate. For example, see Figure 8 where 0 has been deleted and A satisfies property (2) but only C satisfies property (1). A pair of axes through C leaves all other candidates outside of the crosshatched region, while a pair of axes through A results in B being in the crosshatched region. If no candidate is found to satisfy (1), then (2) guarantees that at least one of the candidates has the property that only one of the other candidates will occupy the crosshatched region between the original axes and the axes passing through the selected candidate. To see this, note that whichever candidate is selected to be the new root (say B in quadrant i), then the candidate in quadrant conjugate(i) lies outside of the crosshatched region. In addition, the candidate in a quadrant which is on the same side of an axis, as is B, and to which axis B is closer, lies outside of the crosshatched region.

We are now ready to present our deletion algorithm. It makes use of the properties of the space obtained by the new partition to reduce the number of nodes requiring reinsertion. The algorithm consists of two procedures, ADJ and NEWROOT. Let A be the node to be deleted and let i be the quadrant of the tree rooted at A containing B, the "closest" node which will replace A. Note that no nodes in quadrant conjugate(i) need to be reinserted. Now, separately process the two quadrants adjacent to quadrant i using procedure ADJ.

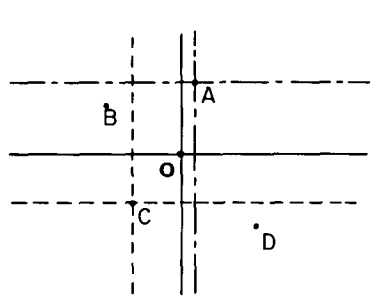


Fig. 8. Example of the insufficiency of (2) for empty crosshatched region.

ADJ: Examine the root of the quadrant, say J. If J lies outside of the crosshatched region, then two subquadrants can automatically remain in the quadrant and need no further processing while the remaining subquadrants are separately processed by a recursive invocation of ADJ. Otherwise, the entire quadrant must be reinserted in the quad tree which was formerly rooted at A.

For example, consider Figure 1 where A is deleted and replaced by B in the first quadrant. J and the subquadrant rooted at K remain in the second quadrant while the subquadrant rooted at L is recursively processed. Eventually, L must be reinserted in the subtree rooted at M. The third quadrant of A (rooted at N) does not require reinsertion. Figure 9 shows the result of the deletion.

Once the nodes in the quadrants adjacent to i have been processed, we must process the nodes in i. Clearly, all of the nodes in subquadrant i of i will retain their position. Bearing this in mind, we apply procedure NEWROOT to the remaining subquadrants of i.

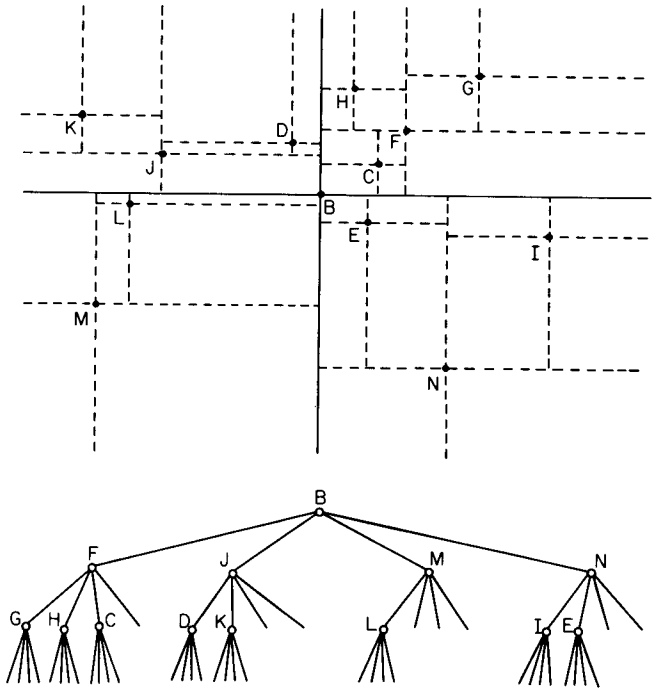
NEWROOT: Apply algorithm ADJ to the subquadrants adjacent to subquadrant i and iteratively reapply NEWROOT to subquadrant conjugate(i). This is done until an empty link in direction conjugate(i) is encountered (i.e., at this point we are at B—the node replacing the deleted node). Now, insert the nodes in the subquadrants adjacent to subquadrant i of the tree rooted at B in the quadrants adjacent to quadrant i of the tree rooted at A. Recall that by virtue of the definition of "closest," subquadrant conjugate(i) of the tree rooted at B is empty. Also subquadrant i of the tree rooted at B replaces subquadrant conjugate(i) of the previous father node of B.

Figure 10 illustrates the subquadrants that are processed by ADJ when node 0 is deleted and the resulting tree is rooted at node 4. For the example of Figure 1, the tree rooted at G is left alone. Trees rooted at H and I are processed by ADJ. Trees rooted at D and E are reinserted in quadrants 2 and 4, respectively. The tree rooted at C replaces B as the son of F in subquadrant 3. Figure 9 shows the result of deleting A from Figure 1.

3. Analysis of the Effect of the Algorithm on the Number of Insertions

In order to measure the efficiency of our algorithm, we analyze the expected number of nodes that must be reinserted. Define a nontrivial subtree to be a subtree with two or more nonempty subtrees. Let $S(n)$ and $Q(n)$ be the expected number of nodes that must be reinserted and the expected nontrivial subtree size, respectively, in a tree of n nodes. $Q(n)$ corresponds to the number of nodes which may need to be examined for reinsertion whenever the root of a nontrivial subtree is deleted. Thus, $Q(n)$ is the cost of deletion when the method of [3] is used. $S(n)$ is the cost when our method is used. Our analysis assumes a quad tree having n nodes that is also complete [5] (i.e., perfect [3]) because, for fairness of comparison, such a configuration minimizes the average cost of the deletion method of [3]—i.e., reinserting the subtree. This can be seen by noting [1, p. 515] that the sum of the subtree sizes, for a given tree T of size n, is TPL(T) + n which is at a minimum when T is a complete quad tree. The effect of the complete quad tree assump-

Fig. 9. Result of deleting node A from the quad tree of Figure 1 and replacing it with node B.



tion is exhibited by the empirical results in the next section. Letting $r(n)$ denote the proportion of nodes that do not require reinsertion when using our algorithm, we have the relationship

$$S(n) = (1 - r(n)) \cdot Q(n) \tag{1}$$

We first derive $Q(n)$. Let v be the maximum level (or depth) in the tree. From the definition of $Q(n)$, all subtrees of size 1 are ignored since no nodes must be reinserted when their root is deleted. In a subtree at level i there are $\sum_{j=0}^{v-i} 4^j$ nodes. There are 4^i such subtrees. Therefore

$$Q(n) = \frac{\sum_{i=0}^{v-1} 4^i \cdot \left(\sum_{j=0}^{v-i} 4^j - 1 \right)}{\sum_{i=0}^{v-1} 4^i}$$

$$= 4 \cdot v \cdot \left(\frac{n}{n-1} \right) + \frac{4 \cdot v}{3 \cdot (n-1)} - \frac{4}{3} \tag{2}$$

Observe that

$$\frac{n-1}{4} = \sum_{i=0}^{v-1} 4^i \tag{3}$$

Allowing n to become large and solving (3) for v enables (2) to be rewritten as

$$Q(n) = 4 \cdot \log_4(\tfrac{3}{4} \cdot n) - \tfrac{4}{3} \tag{4}$$

Next, we derive $r(n)$. Let A and B be the deleted and replacing nodes, respectively, where B satisfies the "closest" criteria of Section 1. Therefore, B is a result of procedure FINDCANDIDATE that satisfies properties (1) and (2). Let the tree be complete with n nodes and v being defined as in (3). Also, assume that the nodes are distributed uniformly throughout the partition of the two-dimensional space rooted at node A.

The contributions to $r(n)$ made by each quadrant of the region associated with the subtree rooted at A are evaluated separately below. Assume without loss of generality that B is in quadrant 1. Let $T = \sum_{i=0}^{v-2} (1/4)^i$.

Contribution from Quadrant 3

No nodes in quadrant 3 need to be reinserted. Hence, this quadrant contributes $\frac{1}{4}$ to $r(n)$.

Contribution from Quadrant 2

There are two possible cases depending on whether or not there is more than one "closest" node such that the crosshatched region contains it.

CASE a. The closest node, say J, to the deleted node, A, in this quadrant lies outside of the crosshatched region (see Figure 1).

By its recursive nature, procedure ADJ eliminates two subquadrants from consideration each time it is invoked. B being the "closest" node assures that the root of the subtree associated with the fourth subquadrant of quadrant 2 also eliminates two of its subquadrants from consideration, etc. Summing over all levels, the first, second, and fourth subquadrants of quadrant 2 contribute

$$\tfrac{1}{4} \cdot (\tfrac{1}{2} + \tfrac{1}{4} \cdot (\tfrac{1}{2} + \tfrac{1}{4} \cdot (\tfrac{1}{2} + \ldots))) = \tfrac{1}{8} \cdot T$$

The third subquadrant of quadrant 2 also makes a contribution to $r(n)$. Since the root of this subquadrant is not one of the nodes returned by FINDCANDIDATE, we let p_{12} denote the probability that the nodes in this

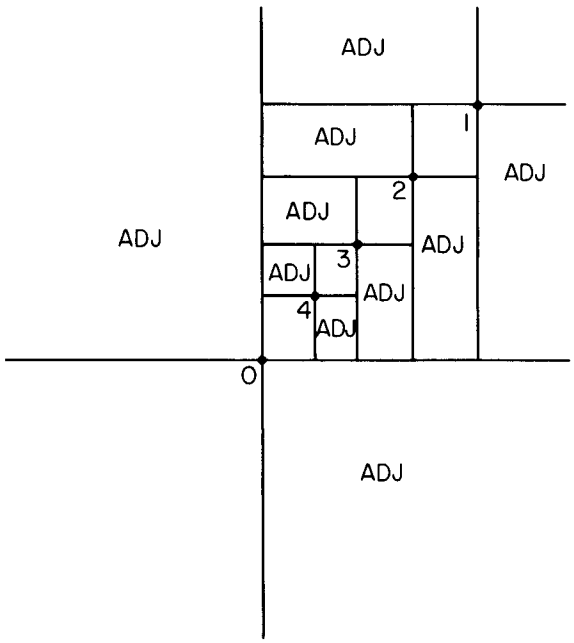


Fig. 10. Subquadrants processed by ADJ when node 0 is deleted and replaced by node 4.

subquadrant (at level 2) will not need to be reinserted. In general, we assume the existence of a sequence of probabilities $\{p_{1i}\}$ such that p_{1i} is the probability that nodes in the third subquadrant at level i, as ADJ is applied recursively, need not be reinserted. Summing over all levels results in

$$\tfrac{1}{4}\cdot\tfrac{1}{4}\cdot(p_{12} + \tfrac{1}{4}\cdot(p_{13} + \tfrac{1}{4}\cdot(p_{14} + \ldots))) \tag{5}$$

Define p_1 so that (5) is equal to

$$\tfrac{1}{4}\cdot\tfrac{1}{4}\cdot(p_1 + \tfrac{1}{4}\cdot(p_1 + \tfrac{1}{4}\cdot(p_1 + \ldots)))$$

p_1 algebraically represents $\{p_{1i}\}$. Note that in terms of p_1, (5) is equal to $1/16\cdot p_1\cdot T$. In general, we refer to p_j as the *representative* of $\{p_{ji}\}$.

Thus, the contribution to $r(n)$ from the second quadrant for Case a is

$$\tfrac{1}{8}\cdot T + \tfrac{1}{16}\cdot p_1\cdot T = \tfrac{1}{16}\cdot(2 + p_1)\cdot T$$

CASE b. The closest node, say C, to the deleted node, A, in this quadrant lies inside the crosshatched area (see Figure 5).

In this case the analysis is similar to that performed for case a with the following modification. Let p_{22} be the probability that the root of this quadrant (at level 2) need not be reinserted, thereby resulting in no need to reinsert the first and second subquadrants. In general, we assume the existence of a sequence of probabilities $\{p_{2i}\}$ such that p_{2i} is the probability that nodes in the first two subquadrants at level i, as ADJ is applied recursively, need not be reinserted. This is somewhat different from the analysis performed for the third subquadrant for case a, since here having to reinsert the root of a subquadrant, say X, means that there is a probability that we need not have to reinsert the first two subquadrants of the fourth subquadrant of X, etc. Summing over all levels results in

$$\tfrac{1}{4}\cdot(p_{22}\cdot\tfrac{1}{2} + \tfrac{1}{4}\cdot(p_{23}\cdot\tfrac{1}{2} + \tfrac{1}{4}\cdot(p_{24} + \ldots))) \tag{6}$$

Once again, define p_2 so that (6) is equal to

$$\tfrac{1}{4}\cdot(p_2\cdot\tfrac{1}{2} + \tfrac{1}{4}\cdot(p_2\cdot\tfrac{1}{2} + \tfrac{1}{4}\cdot(p_2\cdot\tfrac{1}{2} + \ldots)))$$

Thus, p_2 algebraically represents $\{p_{2i}\}$ and (6) is equal to $1/8\cdot p_2\cdot T$.

The contribution of the third subquadrant is obtained in the same way as for Case a. The only difference is that p_3 represents $\{p_{3i}\}$.

Thus, the contribution to $r(n)$ from the second quadrant for Case b is

$$\tfrac{1}{8}\cdot p_2\cdot T + \tfrac{1}{16}\cdot p_3\cdot T = \tfrac{1}{16}\cdot(2 + p_2 + p_3)\cdot T$$

Contribution from Quadrant 4

The analysis is the same as that completed for quadrant 2. Case a yields $1/16\cdot(2 + p_4)\cdot T$, where p_4 is the representative of $\{p_{4i}\}$. Case b yields $1/16\cdot(2 + p_5 + p_6)\cdot T$, where p_5 and p_6 are the representatives of $\{p_{5i}\}$ and $\{p_{6i}\}$, respectively.

Contribution from Quadrant 1

Procedure NEWROOT eliminates one subquadrant (subquadrant 1) from consideration at each iteration. FINDCANDIDATE assures that the third subquadrant will also eliminate its first subquadrant, etc. Summing over all levels, the contribution of the first and third subquadrants to $r(n)$ is

$$\tfrac{1}{4}\cdot(\tfrac{1}{4} + \tfrac{1}{4}\cdot(\tfrac{1}{4} + \tfrac{1}{4}\cdot(\tfrac{1}{4} + \ldots))) = \tfrac{1}{16}\cdot T$$

The second and fourth subquadrants are analyzed in the same way as the third subquadrant for Case a was. With p_7 and p_8 denoting the representatives of probability sequences for these subquadrants, the first quadrant contributes

$$\tfrac{1}{16}\cdot T + \tfrac{1}{16}\cdot p_7\cdot T + \tfrac{1}{16}\cdot p_8\cdot T = \tfrac{1}{16}\cdot(1 + p_7 + p_8)\cdot T$$

Recall from Section 2 that properties (1) and (2) guarantee that at most one of the two quadrants adjacent to the quadrant containing the new root (i.e., 1) will have the property that its "closest" node lies in the crosshatched region. Allowing n to become large yields a value of $\frac{4}{3}$ for T. Thus, if none of the adjacent quadrants have their "closest" node in the crosshatched region, then

$$\begin{aligned} r(n) &= \tfrac{1}{4} + \tfrac{1}{16}\cdot(2 + p_1)\cdot T + \tfrac{1}{16}\cdot(2 + p_4)\cdot T \\ &\quad + \tfrac{1}{16}\cdot(1 + p_7 + p_8)\cdot T \\ &= \tfrac{1}{4} + \tfrac{1}{12}\cdot(5 + p_1 + p_4 + p_7 + p_8) \end{aligned} \tag{7}$$

while when one of the adjacent quadrants has its "closest" node in the crosshatched region, assuming it is quadrant 2, we have

$$\begin{aligned} r(n) &= \tfrac{1}{4} + \tfrac{1}{16}\cdot(2\cdot p_2 + p_3)\cdot T \\ &\quad + \tfrac{1}{16}\cdot(2 + p_4)\cdot T + \tfrac{1}{16}\cdot(1 + p_7 + p_8)\cdot T \\ &= \tfrac{1}{4} + \tfrac{1}{12}\cdot(3 + 2\cdot p_2 + p_3 + p_4 + p_7 + p_8) \end{aligned} \tag{8}$$

Suppose we relax the assumption that the replacing node B satisfies properties (1) and (2). Instead, randomly select one of the candidates as the replacing node. Thus, both of the quadrants adjacent to the quadrant containing the new root (i.e., 1) may have the property that their "closest" node lies in the crosshatched region. In such a case we have

$$\begin{aligned} r(n) &= \tfrac{1}{4} + \tfrac{1}{16}\cdot(2\cdot p_2 + p_3)\cdot T + \tfrac{1}{16}\cdot(2\cdot p_5 + p_6)\cdot T \\ &\quad + \tfrac{1}{16}\cdot(1 + p_7 + p_8)\cdot T \\ &= \tfrac{1}{4} + \tfrac{1}{12}\cdot(1 + 2\cdot p_2 + p_3 + 2\cdot p_5 \\ &\quad + p_6 + p_7 + p_8) \end{aligned} \tag{9}$$

By the nature of the partitioning process, each of the representatives $p_1 - p_8$ approximates the average of a monotonically decreasing sequence. Simulation results followed by curve fitting (discussed in the next section) show that $p_1 - p_8$ can be reasonably approximated by $\frac{1}{2}$. Using this result, we say that for large n, the value of $r(n)$ lies between $\frac{2}{3}$ and $\frac{5}{6}$. Therefore, when properties (1) and (2) are satisfied, the lower bound $S(n)$ is

$$\begin{aligned} S(n) &= (1 - \tfrac{5}{6})\cdot(4\cdot\log_4(\tfrac{3}{4}\cdot n) - \tfrac{4}{3}) \\ &= \tfrac{2}{3}\cdot\log_4(\tfrac{3}{4}\cdot n) - \tfrac{2}{9} \end{aligned}$$

Table I. Observed and Predicted Values of $S(n)$ and $Q(n)$.

		$S(n)$				$Q(n)$			
		closest		random		improved		naive	
size	trials	observed	predicted	observed	predicted	observed	predicted	observed	predicted
25	300	1.39	1.20	3.11	2.40	5.63	5.34	8.69	7.12
50	300	1.73	1.52	3.77	3.03	7.48	6.83	11.0	9.10
100	300	2.02	1.84	4.37	3.68	8.99	8.30	13.1	11.1
200	300	2.38	2.18	5.11	4.36	10.8	9.83	15.6	13.1
500	100	2.69	2.64	5.74	5.28	13.3	11.9	18.9	15.8
1,000	50	2.87	2.96	6.13	5.92	15.4	13.4	21.3	17.8
2,000	25	3.24	3.30	6.64	6.59	16.6	14.8	23.7	19.8

When properties (1) and (2) are not satisfied, the upper bound for $S(n)$ is

$$S(n) = (1 - \tfrac{2}{3}) \cdot (4 \cdot \log_4(\tfrac{3}{4} \cdot n) - \tfrac{4}{3})$$
$$= \tfrac{4}{3} \cdot \log_4(\tfrac{3}{4} \cdot n) - \tfrac{4}{9}$$

In computing $r(n)$ we assumed a very naive version of the algorithm of [3] which reinserts all of the subtrees of the deleted node. Instead, the deleted node can be replaced by one of its subtrees and thus, only the remaining subtrees need to be reinserted. We term this the "improved" version of [3] and when it is used, on the average only $\frac{3}{4}$ as many nodes require reinsertion. Thus, the reinsertion proportion is really $4/3 \cdot (1 - r(n))$. In other words, this proportion ranges between $\frac{2}{9}$ and $\frac{4}{9}$ of that required by the improved version, whereas for the original version the proportion ranges between $\frac{1}{6}$ and $\frac{1}{3}$.

4. Empirical Results

In order to test our theoretical predictions, empirical tests were conducted. Quad trees of varying sizes were created at random with key values ranging from 0 to $2^{31} - 1$. The deletion process was performed for each node in the tree that had two or more subtrees (i.e., the nontrivial cases). Table I contains predicted and observed values for $S(n)$ (i.e., the expected number of insertions) when the replacing node is selected according to properties (1) and (2) (labeled "closest") and when it is chosen at random from the set of candidate nodes (labeled "random"). $Q(n)$ is also given for the naive and improved versions of the algorithm of [3]. As the tree size increases, the predicted and observed values of $S(n)$ are closer in value. This is not surprising because the analysis assumes complete quad trees, whereas this is not true for the simulated trees, and the value of $r(n)$ is an upper bound (curve fitting results in $r(n)$ being 0.829 and 0.637 for the "closest" and "random" cases, respectively). Smaller sized trees result in smaller values of $r(n)$, thereby accounting for the observed values of $S(n)$ exceeding the predicted values for small tree sizes. Also note that the observed values of $Q(n)$ imply that our algorithm results in even greater reductions in the number of nodes requiring reinsertion than indicated by our analysis.

An alternative measure of the amount of work performed by the two deletion algorithms is the number of comparisons made during the process of rearranging the tree once a node has been deleted. This is of interest because the number of reinsertions is not a complete measure of work since the reinsertion of different nodes requires different expenditures of work (i.e., comparison operations) depending on the level of the node in the tree. Using the same tree sizes and trials as in Table I, Figure 11 contains plots for the average number of comparisons made when the improved version of [3] and our algorithm were used. The observed averages for our deletion method result in a proportionality with the log of the tree size.

An additional property that bears watching is the total path length (TPL) of a tree after deletion of a node. This data is significant because it correlates with the effective search time [2, 6]. Our simulations show that "bushiness" (i.e., balance) of the tree has only been slightly affected by the application of our deletion algorithm. For example, compare Figures 2 and 9 which are the results of the application of [3] and our algorithm, respectively. The original TPL was 25 (i.e., Figure 1), whereas the TPL values for Figures 2 and 9 are 22 and

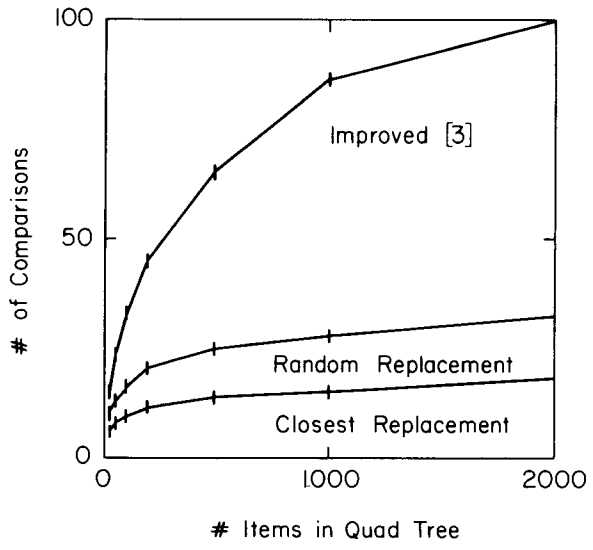


Fig. 11. Average number of comparisons.

Table II. Average Total Path Length (TPL) After Deletion of a Root Node.

size	trials	original	closest	random	method [3]
25	100	68.1	63.5	65.0	71.3
50	100	170.5	163.7	166.6	188.2
100	100	404.1	395.8	401.4	460.7
200	100	942.3	928.8	948.1	1,099
500	100	2,824	2,808	2,841	3,432
1,000	100	6,318	6,313	6,352	7,979
2,000	100	14,127	14,136	14,229	17,590

Table III. Average TPL/Optimal TPL (X) Corresponding to Table II.

size	trials	optimal TPL	original X	closest X	random X	method [3] X
25	100	48	1.4188	1.3229	1.3542	1.4854
50	100	123	1.3862	1.3309	1.3545	1.5301
100	100	288	1.4031	1.3743	1.3938	1.5997
200	100	688	1.3697	1.3500	1.3781	1.5974
500	100	2,047	1.3796	1.3718	1.3879	1.6766
1,000	100	4,547	1.3895	1.3884	1.3970	1.7548
2,000	100	10,182	1.3874	1.3883	1.3975	1.7276

20, respectively. Table II contains a summary of these results when only roots of trees are deleted (instead of all nodes having more than one subtree). Note that as the tree size increases, the TPL increases significantly when the method of [3] is used. For a measure of balance which is independent of the size of the tree we use X, the average TPL divided by the TPL of the optimal tree with the same number of nodes. Table III contains a summary of these results when only roots of trees are deleted. The observed decrease in balance (i.e., an increase in X) results because the simulated algorithms perform reinsertion in a sorted order. However, to do otherwise requires reinserting the tree in random order which is time-consuming. In such a case, the resulting tree has a TPL which is almost identical to that obtained by our methods. This is not surprising since our techniques lead to little variation in the TPL from the original tree which, after all, was created at random. An analysis of the effect of a long sequence of insertions and deletions would be interesting but is beyond the scope of this work (see [5, p. 431]).

5. Concluding Remarks

Two methods for obtaining the replacing node have been discussed: one that chooses a "closest" node at random and one that uses properties (1) and (2). As the tree size increases, the amount of extra work required by the latter is overshadowed by the decrease in subsequent comparison operations. Thus, for large trees the "closest" node should not be selected at random.

The extension of our algorithm to quad trees of higher dimension is worthy of future investigation. In [1] a related data structure termed a *k-d* tree, one inspired by quad trees but fundamentally different in operational costs, is introduced (k denotes the dimension of the space). An interesting problem is to determine if our methods are applicable to *k-d* trees. One of the problems with *k-d* trees in relation to our deletion algorithm is that a node in a *k-d* tree does not partition the space with respect to all key value components as is true for quad trees. Note that the partitioning property makes the quad tree an attractive data structure for applications in which parallelism is feasible (e.g., [7]).

Acknowledgment. Special thanks go to G. Knott for bringing this problem to my attention and for his critical comments and suggestions.

Received 8/78; revised 11/79; accepted 7/80

References

1. Bentley, J.L. Multidimensional binary search trees used for associative searching. *Comm. ACM 18*, 9 (Sept. 1975), 509–517.
2. Bentley, J.L., and Stanat, D.F. Analysis of range searches in quad trees. *Inform. Proc. Lett.* (July 1975) 170–173.
3. Finkel, R.A., and Bentley, J.L. Quad trees: A data structure for retrieval on composite keys. *Acta Informatica*, 4 (1974), 1–9.
4. Knott, G.D. Deletion in binary storage trees. Ph.D. Th., Rep. STAN-CS-75-491, Comptr. Sci. Dept., Stanford Univ., Calif., May 1975.
5. Knuth, D.E. *The Art of Computer Programming, Vol. 3: Sorting and Searching.* Addison-Wesley, Reading, Mass., 1973.
6. Lee, D.T., and Wong, C.K. Worst-case analysis for region and partial region searches in multidimensional binary search trees and balanced quad trees. *Acta Informatica*, 9 (1977), 23–29.
7. Linn, J. A general method for parallel searching. Ph.D. Th., Dept. of Electr. Eng., Stanford Univ., Calif., 1973.
8. Lueker, G.S. A data structure for orthogonal range queries. Proc. 19th Ann. Symp. Foundations Comptr. Sci., Ann Arbor, Mich., 1978, pp. 28–34.