# Dynamic Deferred Data Structuring *

Oct. 1988

by

Y. T. Ching
Institute of Information Science
Academia Sinica
Taipei, Taiwan
Republic of China

K. Mehlhorn
Fachbereich 10, Informatik
Universität der Saarlandes,
Saarbrücken,
Federal Republic of Germany

Abstract: Let $S$ be a set of $n$ reals. We show how to process on-line $r$ membership queries, insertions, and deletions in time $O(r\ log\ (n+r)\ +\ (n+r)\ log\ r)$. This is optimal in the binary comparison model.

Keywords: Data structures, dictionary problem, on-line, weight-balanced trees, deferred data structure.

---

* This research was carried out while the second author was visiting Academia Sinica.

# Dynamic Deferred Data Structuring

Let $S$ be a set of $n$ elements drawn from an ordered universe. We want to process on-line a sequence of $r$ membership queries, insertions, and deletions. If $r \geq n$, then an optimal solution is to sort $S$, construct a balanced search tree for it and then process the sequence for a total cost of $O(n \log n + r \log (n+r))$. If $r \ll n$, then this method is not optimal. For example, for $r = 1$, time $O(n)$ certainly suffices by a simple scan of set $S$. So the question arises whether there is a solution which is optimal for all values of $n$ and $r$. Karp, Motwani, and Raghavan [2] answered this question positively in the static case, i.e., no insertions and deletions are allowed, by giving an $O((n+r) \log min(n,r))$ solution and proving its optimality. They proved similar results for other static query problems, e.g., the membership problem for the convex hull of $n$ points in the plane. In their paper, they posed the challenge to generalize their results to the dynamic situation, i.e., insertions and deletions are also allowed. In this note, we answer this challenge for dictionary problem. For the other problem considered in [2], the question remains open. As in [2], our solution is based on the deferred (or lazy) construction of a balanced search tree for $S$.

Let $S$ be a set of $n$ real numbers. The objective is to process on-line a sequence of $r$ membership queries, insertions and deletions.

**Theorem:** Let $S$ be a set of $n$ reals. Then a sequence of $r$ membership queries, insertions and deletions can be processed in time $O(r \log (n+r) + (n+r) \log r)$ and this is optimal.

Karp, Motwani, and Raghavan proved this theorem for the static situation, i.e., $r$ membership queries are to be processed. Their solution is based on the deferred construction of a perfectly balanced search tree for the set $S$. We use a weight-balanced tree [4,3] instead.

**Definition:** A binary tree is called weight-balanced if for every subtree $T$ the following inequality holds:

$$\frac{1}{4} \leq \frac{|T_l|}{|T|} \leq \frac{3}{4}$$

Here $T_l$ denotes the left subtree of $T$ and $|.|$ denotes the number of leaves of a tree.

Weight-balanced trees can be used as search trees by storing the information in the leaves and using the internal nodes to guide the search. Insertions and deletions are processed by adding or deleting a leaf and then rebalancing the tree along the path of update by rotations and double-rotations, cf. Figure 1.

The following fact is crucial to our approach.

**Fact 1**: Let $T$ be a perfectly balanced tree with $n$ leaves, i.e., $|\frac{|T'|}{2} - |T''|| \leq 1$ for every direct subtree $T''$ of any subtree $T'$ of $T$. Execute an arbitrary sequence of $r$ insertions and deletions on $T$ according to the rebalancing algorithm for weight-balanced trees. Charge $|T_v|$ whenever a (double -) rotation is performed at a node $v$, where $T_v$ is the subtree rooted at $v$. Then the total charge is $O(r \ log \ r)$.

Fact 1 is easily shown using the arguments in Blum and Mehlhorn[1], Willard and Lueker[5]; cf. also Mehlhorn, section III.5.1[4].

A partially expanded weight-balanced tree is obtained from a weight-balanced tree by deleting some of its subtrees and replacing them by a single leaf, called a pseudo-leaf or unexpanded node. All elements stored in the subtree are associated with the pseudo-leaf. For our analysis we need the concept of the potential of a (partially expanded weight-balanced) tree which is defined as follows.

$$potential(T) = \sum_{v, \ pseudo-leaf \ of \ T} depth(v) \cdot size(v)$$

where $size\ (v)$ is the number of elements associated with $v$.

We are now ready for the membership and update algorithms. A membership query for $x$ is processed as in an ordinary weight-balanced tree with one major difference. Whenever the query reaches a pseudo-leaf, the median of the elements associated with the pseudo-leaf is determined and the pseudo-leaf is replaced by a subtree consisting of a node with two pseudo-leaf descendants. Of course, the cost of this operation is proportional to

the increase in potential. In essence, a membership query expands a search path to $x$ if $x \in S$ or to the successor of $x$ if $x \notin S$. In either case, we mark (for the purpose of analysis) the leaf where the search ends. For later use, we state two important properties of our membership algorithm:
1. If $v$ is a pseudo-leaf then $parent(v)$ lies on a path to a marked leaf.
2. The cost of a query is $O(log\ (n+r))$ plus the increase in potential.

Lemma 1: Let $T$ be a partially expanded weight-balanced tree satisfying property 1 with $r$ marked leaves and an underlying weight-balanced tree of $m \leq n+r$ leaves. Then the potential of $T$ is at most $O(m\ log\ r) = O((n+r)\ log\ r)$.
proof: Let $p_i$ be the total contribution of pseudo-leaves of depth $i$. Then $p_i \leq r \cdot m \cdot (\frac{3}{4})^i \cdot i$ since there are at most $r$ pseudo-leaves of depth $i$ each one having a size of at most $m \cdot (\frac{3}{4})^i$. Also $\sum_{i=0}^{d-1} p_i \leq m \cdot d$ for any $d$ since the pseudo-leaves partition the set stored in $T$. Thus

$$\begin{aligned}\sum_i p_i &= \sum_{i<d} p_i + \sum_{i \geq d} p_i \\ &\leq m \cdot d + r \cdot m \cdot \sum_{i \geq d} (\frac{3}{4})^i \cdot i \\ &= O(m \cdot d + r \cdot m \cdot (\frac{3}{4})^d \cdot d)\end{aligned}$$

With $d = \frac{log\ r}{log(\frac{4}{3})}$, the lemma follows.

We turn to insertions and deletions next. We first perform a membership query for the element to be inserted or deleted. In the case of a deletion, we also mark the successor of the element to be deleted. Then we rebalance along the path of insertion or deletion as usually, again with one major exception. If we perform a (double-) rotation at $v$ and a subtree consisting of a single node with two pseudo-leaves arises, then the subtree is collapsed into a single pseudo-leaf, cf. Figure 2. Note that this preserves property 1.

Lemma 2: A (double-) rotation at $v$ causes the potential to decrease by at most $size(v)$.

Proof: immediate from Figure 2.

It is now easy to complete the proof of the theorem. The total cost of the $r$ update operations is $O(r\ log\ (n+r))$ for the search and the rebalancing. Furthermore the total decrease of potential due to rotations caused by updates is $O(r\ log\ r)$ by Fact 1. The total cost of the queries is $O(r\ log\ (n+r))$ plus the total increase in potential. Finally, the total increase of potential is bounded by the final potential (which is at most $(n+r)\ log\ r$ by lemma 1) and the total decrease (which is $O(r\ log\ r)$ by the argument above). So the total cost of the $r$ queries and updates is $O((n+r)\ log\ r + r\ log\ (n+r))$. If $r \geq n$, then our solution is certainly optimal. If $r < n$ then $O((n+r)\ log\ r + r\ log\ (n+r)) = O((n+r)\ log\ min(n,r))$. The optimality now follows from the fact that $\Omega((n+r)\ log\ min(n,r))$ is a lower bound even without updates, cf. Theorem 1 of [2].

# References


[1] Blum, N., K. Mehlhorn, "On the Average Number of Rebalancing Operations in Weight-Balanced Trees", *Theoretical Computer Science* 11, 1980, pp. 303-320.

[2] Karp, R., R. Motwani, and P. Raghavan, "Deferred Data Structuring," *Technical Report UCB/CSD 87-320, University of California, Berkeley*, Dec. 1987.

[3] Mehlhorn, K., "Data Structures and Algorithms 1: Sorting and Searching", Springer, 1984.

[4] Nievergelt, I., E. M. Reingold, "Binary Search Trees of Bounded Balance", *SICOMP.* 2, 1973, pp.33-43.

[5] Willard, D.E., and G.S. Lueker, "Adding range restriction capability to Dynamic Data Structures," *J. ACM*, 32 (1985), pp 597-617

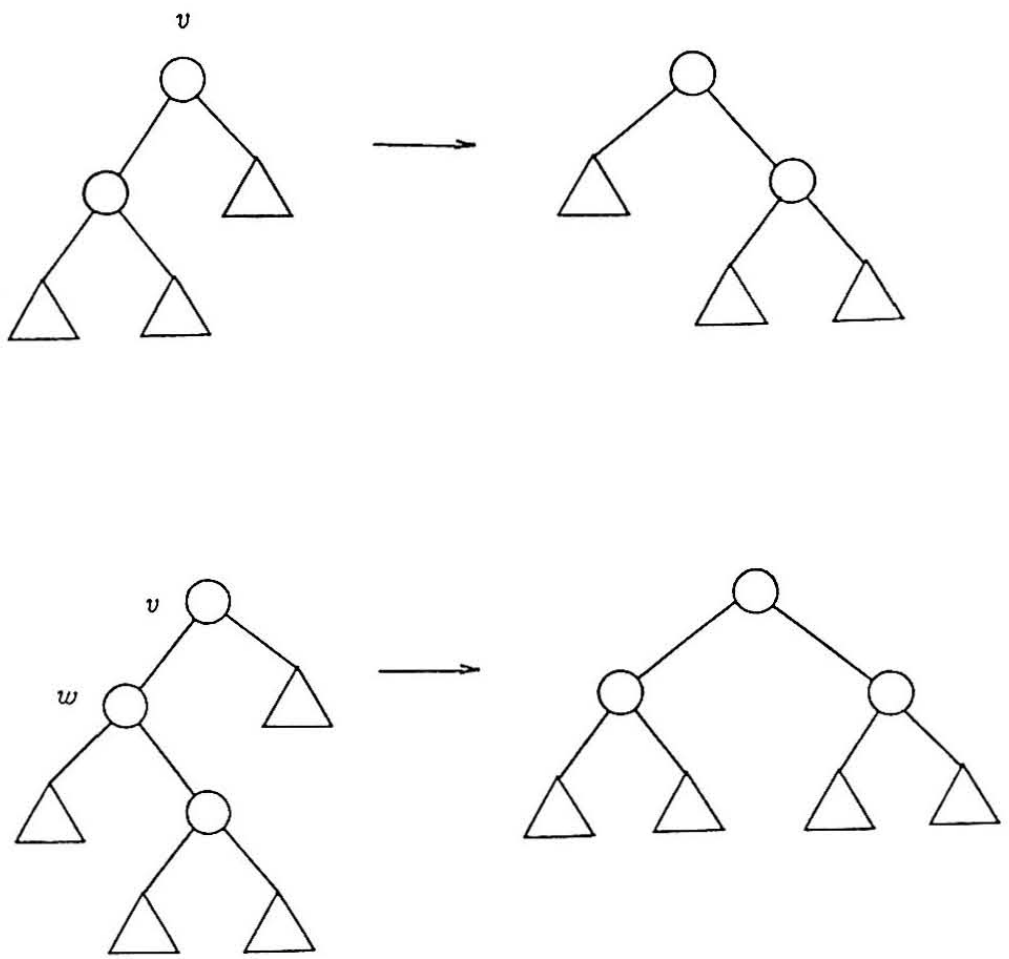


Figure 1: A rotation and a double-rotation at node $v$. Symmetric variants of these operations also exist. The path of update goes through all nodes drawn as circles. A double-rotation at $v$ can be viewed as a rotation at $w$ followed by a rotation at $v$.

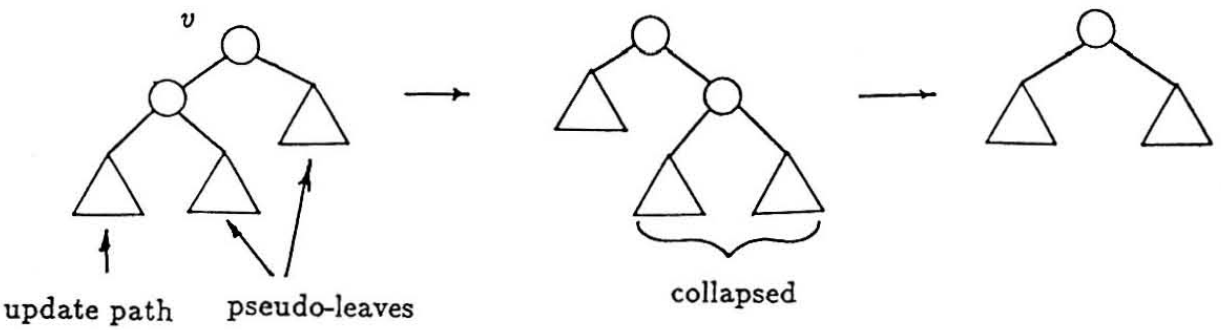


Figure 2: A rotation which causes two pseudo-leaves to be collapsed.