Deletion without Rebalancing in Multiway Search Trees*

Siddhartha Sen¹ and Robert E. Tarjan¹,²

¹ Princeton University
{sssix,ret}@cs.princeton.edu
² HP Laboratories, Palo Alto CA 94304

Abstract. Many database systems that use a B⁺ tree as the underlying data structure do not do rebalancing on deletion. This means that a bad sequence of deletions can create a very unbalanced tree. Yet such databases perform well in practice. Avoidance of rebalancing on deletion has been justified empirically and by average-case analysis, but to our knowledge no worst-case analysis has been done. We do such an analysis. We show that the tree height remains logarithmic in the number of insertions, independent of the number of deletions. Furthermore the amortized time for an insertion or deletion, excluding the search time, is O(1), and nodes are modified by insertions and deletions with a frequency that is exponentially small in their height. The latter results do not hold for standard B⁺ trees. By adding periodic rebuilding of the tree, we obtain a data structure that is theoretically superior to standard B⁺ trees in many ways. We conclude that rebalancing on deletion can be considered harmful.

1 Introduction

Deletion in balanced search trees [2,6,8,10,14,15,18] is a problematic operation. First, if items are stored in the internal nodes of the tree, deletion can require swapping the item to be deleted with its predecessor or successor: this moves the deletion position to the bottom of the tree, where the deletion can be done easily. Second, the rebalancing needed to keep the height of the tree (and the worst-case search time) logarithmic is more complicated than that needed for insertion. Indeed, the original paper on AVL trees [1] did not discuss deletion, and many textbooks neglect it. Third, if operations on the search tree occur in parallel, as in many database systems that use B or B⁺ trees, the synchronization necessary to do rebalancing on deletion reduces the available parallelism [7]. Whereas rebalancing must be done on insertion into a B or B⁺ tree to guarantee correctness (nodes cannot become overfull), it is optional on deletion, since a B or B⁺ tree remains a valid search tree even if it has underfilled nodes.

The first problem with deletion can be overcome by storing the items only in the external nodes of the tree, storing keys in the internal nodes to support search. B⁺ trees [6]

* Research at Princeton University partially supported by NSF grants CCF-0830676 and CCF-0832797 and US-Israel Binational Science Foundation grant 2006204. The information contained herein does not necessarily reflect the opinion or policy of the federal government and no official endorsement should be inferred.
are an example of such a data structure. This takes extra space, but the space penalty
may be worth the benefits. The second and third problems can be addressed by avoiding
rebalancing on deletion. But then the tree need no longer have a height logarithmic
in the number of items. Nevertheless, this method has been used successfully in Berkeley
DB [16, 17], which uses B⁺ trees with underfilled nodes, and in other database systems.

Avoiding rebalancing on deletion has been justified empirically [7, 13, 16, 17] and by
average-case analysis [11, 12], but to our knowledge no one has studied its worst-case
efficiency, perhaps because of the assumption that the worst case, however unlikely,
is terrible. Here we undertake such a study. Perhaps surprisingly, our results provide
ample theoretical justification for avoiding rebalancing on deletion.

One may wonder how this is possible. It is easy to construct an example showing
that the tree height can become arbitrarily large, even if there is only one item left in
the tree [10]. Furthermore the idea of deletion without rebalancing has also been used
in red-black trees, resulting in unforeseen and unfortunate consequences in at least one
application [19]. Nevertheless, it is still possible that the height could remain logarit-
mic in the number of insertions. We show that this is indeed the case. We also show
that the amortized time per insertion or deletion is $O(1)$, and that nodes are affected by
updates with a frequency exponentially small in their heights. These latter results do not
hold for standard B⁺ trees. Thus in some ways deletion with rebalancing is not only not
helpful but actually harmful. Our results provide theoretical support for the design deci-
sion made in Berkeley DB and other database systems to avoid rebalancing on deletion.
In a companion paper [19] we present similar results for balanced binary trees. These
results require careful design of the deletion method; certain natural choices result in
the tree height becoming linear in the number of insertions in the worst case.

The remainder of our paper consists of five sections. In Section 2 we define the B⁻
tree, a relaxed form of B⁺ tree in which deletions are done without rebalancing. B⁻ trees
are essentially those used in Berkeley DB. We describe how to do searches, insertions,
and deletions in such trees. Insertions require node-splitting, which can be done either
top-down or bottom-up; we describe both methods. Deletions require only deletion of
empty nodes. In Section 3 we analyze B⁻ trees. We show that the height, and hence
the search time, is $O(\log_b m)$, where $m$ is the total number of insertions and $b$ is the
maximum node degree. This bound is independent of the number of deletions. We also
show that an insertion or deletion takes $O(1)$ amortized time in addition to a search, and
that nodes are modified by insertions and deletions with a frequency that is exponentially
small in their height. (These results require $b > 3$ if node-splitting is top-down.)

In Section 4 we discuss how and when to rebuild the tree. Such rebuilding eliminates
two drawbacks of B⁻ trees: it keeps the space used proportional to the number of items
in the tree, and it keeps the height logarithmic in the number of items. If rebuilding
is done appropriately often, the amortized rebuilding time is $O(\epsilon)$ per insertion for an
arbitrarily small positive constant $\epsilon$. In Section 5 we sketch how to do rebalancing with
deletion while retaining our inverse exponential bounds on node updates. Section 6 con-
tains some concluding remarks, including a comparison between the case of multiway
trees and that of binary trees.