Abstract. We consider the problem of maintaining a binary search tree that minimizes the average access cost with respect to randomly generated requests. We analyze scenarios, in which the accesses are generated according to a vector of probabilities, which is fixed but unknown.

In this paper we devise policies for modifying the tree structure dynamically, using rotations of accessed elements towards the root. Our aim is to produce good approximations of the optimal order of the tree, while minimizing the amount of rotations.

We first introduce the Move Once (MO) rule, under which the average access cost to the tree is shown to equal the average access cost under the commonly used Move to the Root (MTR), at each reference. The advantage of MO over other rules is that MO relocates each of the items in the tree at most once. Consequently, modifying the tree by the MO rule results in $O(n \log n)$ rotations (with $n$ the number of items) for any infinite sequence of accesses.

Then we propose to combine the MO with the usage of counters (accumulating the reference history for each item), that provide approximations of the reference probabilities. We show, that for any $\delta, \alpha > 0$, this rule (which we call MOUCS) approaches the optimal cost to within a difference of $\delta$ with probability higher than $1 - \alpha$, after a number of accesses, which is linear in $n$ times $1/\alpha$ times $1/\delta^2$.

1 Introduction

The Binary Search Tree (BST) is commonly used for storing tables and lists. The advantage of a tree is that it allows an efficient search of the table (or list). Typically, the search is most efficient when the tree is kept as balanced as possible, and when frequently accessed elements are close to the root. We study heuristics which maintain a BST in a nearly optimal form.

* e-mail: hofri@cs.uh.edu.

** Currently at IBM T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598. e-mail: hadas@watson.ibm.com. Author supported in part by the Technion V.P.R. Fund – E. and J. Bishop Research Fund and by the Fund for the Promotion of Research at the Technion.
The scenario considered has a set of $n$ records in random storage, $L = \{R_1, \ldots, R_n\}$. The record $R_i$ is uniquely identified by the key $K_i$, for $1 \leq i \leq n$, and the keys satisfy a total order. The set is maintained as a BST. The records are accessed according to a multinomial distribution driven by the fixed Reference Probability Vector (RPV): $\hat{p} = (p_1, \ldots, p_n)$. Thus, $R_i$ may be requested at any stage with the same probability $p_i$, independently of previous requests and the state of the tree – and in particular of the location of this $R_i$. This is the so-called independent reference model (IRM). Since the RPV and $L$ are constant, the passage of time is realized by the sequence of references. There is no other notion of time in the model.

Each reference requires a search for a record in the tree. The cost of a single access is defined as the number of key comparisons needed to locate the specified record.

The order by which the records are initially inserted into the tree is assumed to be random (with equal probability over all possible permutations). Different initial insertion sequences may result in different trees, usually with different expected access cost.

The access probabilities listed in the RPV $\hat{p}$ are assumed unknown. Were they known, we could restructure the tree, using a dynamic programming approach to provide the smallest possible expected cost. Since the RPV is constant, so would be the optimal structure. With $\hat{p}$ unknown, we are reduced to looking at policies that use the accumulating reference history to adapt the tree structure with the goal of rearranging the records so that the expected access cost is minimized.

The reorganization process incurs a cost as well: the manipulations performed on the tree when its structure is modified. The only operations used for this are rotations, operations that exchange the ‘rotated’ node with its parent, while maintaining the key-order in the tree. Figure 1 shows the tree modifications that result. Note that the inverse of the rotation operation is a rotation as well. The cost of the reorganization is defined as the number of rotations, since each rotation requires essentially the same computing time.

![Diagram](image)

**Fig. 1.** Single left-child and right-child rotations that reflect $(A) < K_a < (B) < K_b < (C)$