WEIGHTED MULTIDIMENSIONAL B-TREES USED AS NEARLY OPTIMAL DYNAMIC DICTIONARIES

H.P. Kriegel
Informatik VI, Universität Dortmund
D-4600 Dortmund 50, W.Germany

V.K. Vaishnavi
Department of Computer Science
Ohio University, Athens, Ohio 45701, USA

Abstract:
We suggest weighted (k+1)B-trees, k≥1, for weighted dynamic dictionaries of items with k-dimensional keys. It is shown that they compare favorably with a data structure recently introduced for the same application.

1. Introduction

Recently, Bent, Sleator and Tarjan [2] (1980) suggested biased 2-3 trees as a data structure for weighted dynamic dictionaries and Sleator and Tarjan [8] (1981) used biased 2-3 trees for solving various network flow problems. A weighted dynamic dictionary, dynamic dictionary for short, is an abstract data structure that stores a collection of items, each of which has a key, a weight, and possibly other information (depending on the application). The keys are drawn from a totally ordered set. Bent et al. only consider the case of one-dimensional keys. The weights are positive real numbers, presumably representing the relative importance of the items. Specially, they can be considered as the number of accesses to the item. We will refer to the weight of K, denoted w(K), instead of referring to the weight of the item with key K. Let W be the total weight of all items in dictionary D, \[ W = \sum_{K \in D} w(K). \]

The following operations are defined on dynamic dictionaries:
1. Given a key K, ACCESS the item with key K.
2. INSERT a new item in the dictionary.
3. Given a key K, DELETE the item with key K from the dictionary.
4. Given a key K and a real number δ, PROMOTE the key to the weight w(K) + δ.
5. Given a key K and a real number δ, DEMOTE the key to the weight w(K) - δ, provided the result is still positive.
6. If all keys in dictionary D₁ are smaller than all keys in dictionary D₂, CONCATENATE D₁ and D₂ to a new dictionary D containing all items of D₁ and D₂.

7. Given a key K, SPLIT the dictionary into three parts: a new dictionary containing the items with keys less than K, the item with key K and a new dictionary containing the items with keys greater than K.

Mehlhorn [6,7] (1978, 1979) has suggested an implementation for dynamic dictionaries called D-trees in which it is possible to achieve logarithmic behavior per operation for ACCESS, INSERT, DELETE, PROMOTE and DEMOTE. The biased 2-3 trees are simpler than the D-trees and use only linear space. The price that has to be paid for simplicity is that the running time must be amortized over a sequence of operations to achieve logarithmic behavior. Very recently, Güting and Kriegel [4] (1981) have presented the weighted 2B-tree of order d, d≥1, with the following worst case time complexities per operation:

1. $O(\log_{d+1} w(K))$ time for ACCESS and PROMOTE.
2. $O(\log_{d+1} w)$ time for INSERT and DELETE.
3. $O(\log_{d+1} w(K) - \delta))$ time for DEMOTE.

The weighted 2B-tree has a similar simplicity as the biased 2-3 tree and uses only linear space. Yet its advantage is that it guarantees logarithmic behavior per operation and not amortized logarithmic behavior. Further advantages are:

1. The structure is suitable for external stores (d≥1).
2. The structure generalizes naturally to k-dimensional keys (for weighted (k+1)B-trees a factor of (k-1) is added to the time complexities).

Thus weighted 2B-trees will be an efficient implementation for dynamic dictionaries if algorithms for the operations CONCATENATE and SPLIT with logarithmic behavior per operation will be provided. This will be done in sections 3 and 4.

2. Weighted 2B-trees

In this section we will shortly review the structure of weighted 2B-trees. For storing one-dimensional keys with weights a twodimensional B-tree (see Güting and Kriegel [3] (1980)) of some order d, d≥1, is used. The keys are stored in the first dimension level (counted from top). In a usual 2B-tree, the EQSON pointer of key K points to a subtree storing all second components of keys with common first component K. This EQSON subtree of K does not exist in a weighted 2B-tree. Although it is only virtual, by its own height which depends on w(K) it determines the height of key K.

Example 1:
Figure 1 shows a weighted 2B-tree of order 1 for the pairs $(K_i, w(K_i))$, 1≤i≤5: