Pergamon

Computers Math. Applic. Vol. 30, No. 7, pp. 47–66, 1995

0898-1221/95 $9.50 + 0.00

0898-1221(95)00125-5

Space Saving Generalization of B-Trees with 2/3 Utilization

K. V. SHVACHKO
Lesnoj 3, Apt. 21, Pereslavl-Zalessky, 152140, Russia
shv@infos.botik.yaroslavl.su

(Received December 1994; accepted January 1995)

Abstract—The paper studies balanced trees with variable length records. It generalizes the concept of B-tree with unfixed key length introduced in [1] and $S(1)$-tree of [2]. The main property of the new trees, called $S(b)$-trees, is their local incompressibility. That is, any sequence consisting of $b + 1$ neighboring nodes of the tree cannot be compressed into a b well formed node. The case of $S(2)$-trees is studied in detail. For these trees, $2/3 - \varepsilon$ utilization lower bound is proven, where ε is inversely proportional to the tree branching. Logarithmic running time algorithms for search, insertion, and deletion are presented.

Keywords—Data structures, Maintaining dictionaries, Balanced trees, B-trees, Variable-length records.

1. INTRODUCTION

The problem of maintaining dynamic dictionaries is a classical problem of the theory of data structures. As usual, "dynamic dictionary" means a data structure for storing the dictionary elements, called keys, together with algorithms of access to, insertion, and deletion of a key.

Several tree data structures have been developed that provide for efficient solutions to the problem. It is well known that access in a tree-like structure requires logarithmically many comparisons of keys [3]. In order to achieve this lower bound, the balanced trees should be used. However, fully balanced trees occur rarely. Therefore, so called weak balance conditions were considered that lead to approximately equal access time for different tree keys rather than exactly equal, as in case of the fully balanced trees.

Probably, the first data structure for the problem was studied by G. M. Adelson-Velskii and E. M. Landis [4,5]. The approach is based on binary trees that satisfy the following balance condition. For any tree node, the difference between the heights of its two child subtrees is at most 1. These trees, called AVL-trees, allow of logarithmic time algorithms of search, insertion and deletion, and are appropriate for maintaining dictionaries in the internal memory.

Another variant of weak balance condition is used in 2-3-trees, which for the first time were examined by J. Hopcroft in 1970 (unpublished, see [6]). In contrast to AVL-trees, all leaf nodes of a 2-3-tree are located at the same level of the tree. Balancing here is achieved by varying the out-degrees of the internal nodes, which can be either 2 or 3 [7,8]. It is interesting to note that any 2-3-tree can be transformed into an AVL-tree.

B-trees, proposed by R. Bayer [9,10], generalize 2-3-trees. They have the same regular structure, but the out-degree of the tree internal nodes may range between $q + 1$ and $2q + 1$, for

I am grateful to A. Semenov for permanent attention to my work and helpful remarks, to A. Stolboushkin, and to H. Reiser for reading and commenting on a draft of the paper.

a fixed parameter q, called the tree order. From a practical viewpoint B-trees are suitable for maintaining dictionaries in external memory. While a number of variants of B-trees have been examined [3,11–13], all of them are based on the same idea of counting the number of keys in (equivalently, the number of children of) a node.

What is important for us is that with at least q and at most $2q$ keys in each node, any n-vertex B-tree is going to hold no less than qn, and no more than $2qn$ keys. In other words, we say that the tree utilization is lower–bounded by $1/2$, where utilization is the ratio of the total number of keys, contained in the tree, to the maximal number $2qn$ of keys that can contain a B-tree with n nodes.

The disadvantages of B-trees have been widely discussed (see, e.g., [1,11,14]) in the context of this space lower bound. The bottom line is that B-trees utilize memory well only when the keys are of (almost) identical length, while when the keys can differ greatly from each other, they lead to an exhaustive waste of memory.

It is quite clear that in order to store variable length keys efficiently in a balanced tree, the nodes of the tree should restrict the total sum of lengths of keys in a node rather than the number of keys per node, and that the balance conditions should be formulated in adequate terms.

Probably for the first time, this idea was mentioned in [11] (with a reference to an unpublished result of T. H. Martin). Martin's idea leads to a modification of B-trees with the balance condition very similar to that of B-trees, which allows to keep the algorithms almost unchanged. The idea was precisely formulated, developed, and analyzed in [1].

In [1], it is also suggested a new tree data structure, called B-trees, with unfixed key length, which differs from the modification above in that the balance condition is formulated not locally for each tree node, but in terms of pairs of neighboring nodes. It provides high density of a tree in a whole even if some of the nodes are almost empty. Such nodes in the tree are "balanced" with their neighbors, which must be almost full in this case.

It was proven that utilization of any B-tree with unfixed key length exceeds $1/4$. Utilization for the new trees is different from the utilization of usual B-trees. In our case, utilization is the ratio of the total length of keys, contained in the tree, to the maximal length np of keys that can contain such a tree with n nodes.

Utilization computed for B-trees in that way cannot be bounded by any constant greater than 0.

The next step towards exploration of balanced trees with variable length keys was done in [2], where the notion of B-trees with unfixed key length was generalized by introducing $S(1)$-trees. A $S(1)$-tree in addition to the balance condition of the unfixed key length trees requires that the number of keys in the tree node is lower bounded by some constant q, called the tree order. This requirement is resolved in high branching of the tree internal nodes, and raises the utilization of the tree. We proved the $1/2 - \varepsilon$ utilization lower bound for these trees, where ε is inversely proportional to the tree order. Logarithmic running time bounds remain unchanged compared to the case of B-trees with unfixed key length.

$S(1)$-trees appeared to be more efficient than traditional B-trees for many applications. Historically for the first time $S(1)$-trees were implemented in the Starset programming language for representing set data aggregates [15].

This paper presents one more (and, probably, the last) generalizing idea. We introduce here the notion of $S(b)$-tree (read as sweep-be-tree) where b is a parameter, called local parameter. The main property of $S(b)$-trees is their incompressibility. Informally, incompressibility means that any nonroot node of a $S(b)$-tree being composed with any b of its nearest neighbors cannot be compressed into b proper nodes.

We give the general definition of $S(b)$-trees, and study the case of $b = 2$ in detail. This case is of great importance, as $S(2)$-trees qualitatively differ from $S(1)$-trees. They give rather complete characterization of the general case of $S(b)$-trees, being simple enough at the same time. We

prove the $2/3 - \varepsilon$ utilization lower bound for $S(2)$-trees, and present logarithmic running time algorithms of search, insertion, and deletion for them.

The author's ongoing research is to study the general case of $S(b)$-trees for arbitrary parameter b.

2. BASIC DEFINITIONS

2.1. Trees

DEFINITION 2.1. *Let K be a finite set of elements called keys. Define tree inductively:*

(1) *Object Λ, called empty tree, is a tree;*
(2) *If $k_1, \ldots, k_m \in K$ and $T_0, T_1, \ldots, T_m$ are trees, then tuple $\langle T_0, k_1, T_1, \ldots, k_m, T_m \rangle$ is also a tree.*

By *subtree* of nonempty tree

$$T = \langle T_0, k_1, T_1, \ldots, k_m, T_m \rangle$$

we mean T itself and any subtree of any tree T_i $(0 \leq i \leq m)$.

If S and R are subtrees of T, and R is a subtree of S, then R is called a *descendant* of S in T, and S is called an *ancestor* of R in T. If S and R are subtrees of T, such that $S = \langle S_0, l_1, S_1, \ldots, l_m, S_m \rangle$ and $S_i = R$ for some i, then R is called the i^{th} *direct descendant of* S, and S is called the *direct ancestor of* R in T.

The usual *graph representation* is very natural for these kinds of trees. For each tree T a (labeled) graph $G(T) = \langle V(T), E(T) \rangle$ that represents it consists of the vertex set $V(T)$, that is the set of all nonempty subtrees of T, and the set of edges $E(T)$, which is defined in the following way. If S and R are nonempty subtrees of T, such that R is the i^{th} direct descendant of S, then the pair $\langle S, R \rangle$ is an edge labeled by number i. Each vertex $S = \langle S_0, l_1, S_1, \ldots, l_m, S_m \rangle$ in the tree is labeled by the sequence of keys $\langle l_1, \ldots, l_m \rangle$. The number m of keys in the sequence is called an *order of the vertex*.

The *out-degree* of a vertex is the number of its **nonempty** direct descendants. Vertex S is called a *leaf* if its out-degree is 0. Vertex $S = \langle S_0, l_1, S_1, \ldots, l_m, S_m \rangle$ is an *internal vertex* if its out-degree equals $m + 1$. Vertex T is called the *root* of tree T.

A sequence of natural numbers $(i_1, \ldots, i_n)$ is called a *path of length* n in a tree T, iff there exists a sequence of nonempty subtrees $S_0, S_1, \ldots, S_n$ of T, such that $S_0 = T$ and each S_j is the i_j^{th} direct descendant of S_{j-1} $(j = 1, \ldots, n)$. We say in this case that the path leads to the vertex S_n, and length n is called the *level* of S_n in T. When n is small, then the level is called *low*, and when n is large, then the level is called *high*.

Height of a tree is its maximal path length.

We say that a vertex F is a *common ancestor* of vertices S and R, iff F is an ancestor for both S and R. F is said to be a *nearest common ancestor* of vertices S and R, if it is their highest common ancestor.

For each tree $T = \langle T_0, k_1, T_1, \ldots, k_m, T_m \rangle$, let $k(T)$ denote the set of keys contained in the root vertex of T:

$$k(\Lambda) = \emptyset, \qquad k(T) = \{k_1, \ldots, k_m\}.$$

And let $K(T)$ denote the total set of keys contained in tree T:

$$K(\Lambda) = \emptyset, \qquad K(T) = k(T) \cup K(T_0) \cup \cdots \cup K(T_m).$$

Thus, for each vertex S of a tree, $K(S)$ denotes the set of keys contained in the subtree S, and $k(S)$ is the set of keys from $K(S)$ that does not belong to any proper subtree of S.

2.2. Structured Trees, B-Trees

An *ordered set of keys* is a pair $(K, \ll)$, where K is a finite set of keys, and $\ll$ is a linear order on K.

We will deal further only with trees of the following special form.

DEFINITION 2.2. *Let $(K, \ll)$ be an ordered key set. A tree T is called structured iff*

(1) *Each vertex of T is either an internal vertex or a leaf,*
(2) *All paths in T from the root to leaves have equal length,*
(3) *For each vertex $S = \langle S_0, l_1, S_1, \ldots, l_m, S_m \rangle$ of tree T*

$$l_1 \ll \cdots \ll l_m, \quad \text{and}$$
$$\forall i \, (1 \le i \le m \Rightarrow (\forall k \in K(S_{i-1})) \; (\forall k' \in K(S_i)) \; (k \ll l_i \ll k')).$$

The *order of a tree* T is a natural number q, such that for each nonroot vertex of the tree its order is at least q. Note, that if the order of a structured tree is q, then the out-degree of each of its internal nodes is greater than q. By Definition 2.1, an order of any tree is at least 1.

DEFINITION 2.3. *Let $(K, \ll)$ be an ordered key set. Let $q > 0$ be a natural number. A structured tree T of order q is called B-tree of order q iff for each of its vertices S its order*

$$|k(S)| \le 2q.$$

The *utilization* of an n-vertex B-tree T of order q is given by the ratio

$$\Delta(T) = \frac{|K(T)|}{2qn}.$$

B-trees are usually used as a data structure for the problem of *maintaining dynamic dictionaries* with respect to the three basic operations of *search*, *insertion*, and *deletion* of a key. It is well known for B-trees that the following proposition holds.

PROPOSITION 2.1. *If T is an n-vertex B-tree of order q, then*

(1) $\Delta(T) > 1/2 - 1/(2n)$,
(2) *search for, insertion, and deletion of a key in T can be performed in time $O(\log n)$.*

The first property is a consequence of Definition 2.3. The classical algorithms for the basic operations are well known and can be found in [6,11–13].

2.3. $DS(b)$-Trees

A *weighted ordered set of keys* is a triplet of $(K, \ll, \mu)$, where $(K, \ll)$ is an ordered key set, and μ is a *weight function* that maps each key $k \in K$ to its weight $\mu(k)$, which is a positive natural number.

Let us denote $\mu_{\max}(K) = \max\{\mu(K) \mid k \in K\}$.

Let $E \subseteq K$. The *weight of set* E is given by

$$\mu(E) = \sum_{k \in K} \mu(k).$$

If S is a vertex of a tree T, then its weight is $\mu(S) = \mu(k(S))$. The *complete weight of tree* T is $M(T) = \mu(K(T))$.

The *rank of a tree* is a natural number p, such that for each vertex S of the tree its weight is $\mu(S) \le p$.

Consider a structured tree T. And let us introduce a *neighboring relation* for the vertices of the tree in the following way. Let L and R be vertices of the same level of tree T. Consider a set of keys $I = K(L) \cup K(R)$. Then vertex L is said to be a *left neighbor* of vertex R, and R is said to be a *right neighbor* of L, iff there exists a **unique** key k in $K(T) \setminus I$, that satisfies the following property:

$$(\forall l \in K(L))\ (\forall r \in K(R))\ (l \ll k \ll r).$$

Key k is called the *delimiting key* for neighboring nodes L and R. We say also that k *separates* the neighbors.

Note that the delimiting key for any pair of neighboring nodes belongs to the nearest common ancestor of the neighbors.

A sequence $S_0, k_1, S_1, \ldots, k_m, S_m$ of vertices and keys of a tree T is called a *sweep* iff each pair S_{i-1}, S_i of nodes $(i = 1, \ldots, m)$ of the sweep is a pair of neighbors in the tree, and k_i is their delimiting key. The number m of delimiting keys in the sequence is called the *length of the sweep.*

A sweep $S_0, k_1, S_1, \ldots, k_m, S_m$ of length m of a rank p structured tree T is said to be *dense* iff the following inequality holds:

$$\mu(S_0) + \mu(k_1) + \mu(S_1) + \cdots + \mu(k_m) + \mu(S_m) > mp.$$

A structured tree T of rank p is said to be *b-locally dense* iff each of its sweeps of length b is dense. The number b in this case is called the *locality parameter.*

DEFINITION 2.4. *Let $(K, \ll, \mu)$ be a weighted ordered set of keys. A structured b-locally dense tree T of order q and rank p is called $DS(b)$-tree of order q and rank p iff its parameters b, q and p are related by the following inequalities:*

$$b > 0, \qquad q \geq b, \qquad p \geq 2q\mu_{\max}(K).$$

Let $DS(b, q, p)$ denote the class of $DS(b)$-trees of order q and rank p. Note that if $q_1 \geq q_2$, then $DS(b, q_1, p) \subseteq DS(b, q_2, p)$. The class $DS(b, b, p)$ will be denoted by $DS(b, p)$, and trees of this class will be called $DS(b)$-trees of rank p.

2.4. $S(b)$-Trees

Let a collection $S_0, S_1, \ldots, S_m$ of trees and a collection $k_1, \ldots, k_m$ of keys be given. According to Definition 2.1, we can construct from these two collections a new tree $S = \langle S_0, k_1, S_1, \ldots, k_m, S_m \rangle$, which is graphically represented by the root node, marked with the sequence $k_1, \ldots, k_m$, with $m + 1$ outgoing edges, connecting it with the root nodes of $S_0, S_1, \ldots, S_m$.

We need one more tree constructor, which given two collections $S_0, S_1, \ldots, S_m$ of trees and $k_1, \ldots, k_m$ of keys builds a new tree denoted by $S = [S_0, k_1, S_1, \ldots, k_m, S_m]$. The new tree is obtained by putting together all of the root nodes S_i separated by the keys k_i in one new root node. Formally, if $S_i = \langle S_{i0}, l_{i1}, S_{i1}, \ldots \rangle$ for $i = 0, \ldots, m$, then

$$[S_0, k_1, S_1, \ldots, k_m, S_m] = \langle S_{00}, l_{01}, S_{01}, \ldots, k_1, S_{10}, l_{11}, S_{11}, \ldots, k_m, S_{m0}, l_{m1}, S_{m1}, \ldots \rangle .$$

Particularly, for two vertices $[\langle L_0, l_1, L_1 \rangle, k, \langle R_0, r_1, R_1 \rangle] = \langle L_0, l_1, L_1, k, R_0, r_1, R_1 \rangle$.

A sequence $S_0, k_1, S_1, \ldots, k_m, S_m$ is called an $(m + 1)$-partition of a vertex S iff

$$S = [S_0, k_1, S_1, \ldots, k_m, S_m] .$$

Every $(m + 1)$-partition of a vertex S is determined by the sequence, consisting of its m keys, which uniquely specifies the respective sequence of $m + 1$ nodes.

An $(m+1)$-partition of a vertex S is called *proper with respect to parameters p and q*, or *(p,q)-proper*, iff for all $i = 0, 1, \ldots, m$ the following holds:

$$\mu(S_i) \le p \quad \text{and} \quad |k(S_i)| \ge q.$$

When parameters p and q are clear from the context, we will call such a partition simply *proper*.

A sweep $S_0, k_1, S_1, \ldots, k_m, S_m$ of length m of a structured tree T of order q and rank p is said to be *incompressible* (with respect to the parameters p and q), iff there does not exist any proper m-partition of the vertex $S = [S_0, k_1, S_1, \ldots, k_m, S_m]$, that is, if it is not possible to partition this vertex into m "proper" nodes.

Finally, a structured tree T of order q and rank p is called *b-locally incompressible*, iff each of its sweeps of length b is incompressible.

DEFINITION 2.5. *Let $(K, \ll, \mu)$ be a weighted ordered set of keys. A structured b-locally incompressible tree T of order q and rank p is called $S(b)$-tree of order q and rank p iff its parameters b, q and p are related by the following inequalities:*

$$b > 0, \qquad q \ge b, \qquad p \ge 2q\mu_{\max}(K).$$

Similarly to the $DS(b)$-trees, we introduce also the respective tree classes, denoted by $S(b,q,p)$ and $S(b,p)$.

3. COMPARING THE DATA MODELS

In [2], an analysis of the $S(1)$-tree data model was given. In this paper, the tree data structures with the locality parameter $b = 2$ are analyzed.

Particularly, in [2] it was shown that $S(1)$-trees generalize the notion of B-trees in the following sense.

EXAMPLE 3.1. Consider the weight function μ_0 that identically equals 1 on K. Then any B-tree of order q will be a $DS(1)$-tree of rank $2q$ with the weight function μ_0. Indeed,

(1) $\mu_0(S) = |k(S)| \le 2q$, and
(2) for any pair of neighboring nodes L and R with a delimiting key k

$$\mu_0(L) + \mu_0(k) + \mu_0(R) = |k(L)| + 1 + |k(R)| \ge q + 1 + q > 2q.$$

The reverse is not true. Consider a pair of neighbors L and R of a $DS(1)$-tree whose weights are $q-1$ and $q+1$, respectively. The sum $\mu_0(L) + \mu_0(k) + \mu(R) > 2q$ for any key k then, but $|k(L)| < q$. Thus, the $S(1)$-tree is not a B-tree.

It is easy to see that $DS(1)$-trees are equivalent to $S(1)$-trees.

PROPOSITION 3.1. $S(1,q,p) = DS(1,q,p)$.

PROOF. A structured tree T of order q and rank p is 1-locally dense if and only if it is 1-locally incompressible. ∎

The difference between density and incompressibility manifests itself when the locality parameter b is at least 2.

LEMMA 3.2. *Let S be a vertex such that $|k(S)| > 2q$, and none of its 2-partitions is (p,q)-proper. Then $\mu(S) > 2p$.*

PROOF. Let us assume that $\mu(S) \le 2p$. It will be shown then that there exists a (p,q)-proper 2-partition of vertex S.

As we mentioned above, each key $d \in k(S)$ determines a 2-partition of S, which is denoted by $S^-(d), d, S^+(d)$. That is, $[S^-(d), d, S^+(d)] = S$.

Consider a pair of sets

$$D = \left\{d \in k(S) \mid \left|k\left(S^-(d)\right)\right| \geq q \text{ and } \left|k\left(S^+(d)\right)\right| \geq q\right\},$$
$$E = \left\{d \in k(S) \mid \mu\left(S^-(d)\right) \leq p \text{ and } \mu\left(S^+(d)\right) \leq p\right\}.$$

The sets are not empty, since $|k(S)| \geq 2q+1$ by hypothesis of the lemma, and $\mu(S) \leq 2p$ by our assumption. We need to prove that the intersection of the sets is also not empty.

Suppose that $D \cap E = \emptyset$. Let $S = \langle S_0, k_1, S_1, \ldots, k_m, S_m\rangle$, where $m > 2q$. Then, by definition, the set $D = \{k_{q+1}, \ldots, k_{m-q}\}$, and the set E forms a single contiguous subinterval of the linearized key set $k(S)$, that is

$$\forall x, y, z\,(x < y < z \text{ and } k_x \in E \text{ and } k_z \in E \Rightarrow k_y \in E).$$

Therefore, if $D \cap E = \emptyset$, then either $E \subseteq \{k_1, \ldots, k_q\}$ or $E \subseteq \{k_{m-q+1}, \ldots, k_m\}$.

Consider the first case. Here, for the key k_{q+1} by Definition 2.5, we have

$$\mu\left(S^-\left(k_{q+1}\right)\right) \leq q\mu_{\max}(K) \leq p.$$

Since by the definition of the set E for any key $k \in E$ we have $\mu(S^+(k)) \leq p$, then

$$\mu\left(S^+\left(k_{q+1}\right)\right) \leq \mu\left(S^+(k)\right) \leq p.$$

This means that $k_{q+1} \in E$ contrary to the assumption that the sets D and E do not intersect.

Similarly in the case when $E \subseteq \{k_{m-q+1}, \ldots, k_m\}$, it turns out that key $k_{m-q} \in E$ is at the same time an element of the set D. This contradicts the assumption again.

Thus we can conclude that $D \cap E \neq \emptyset$, and therefore a (p,q)-proper 2-partition exists. This contradicts the hypothesis of the lemma. That is, the initial assumption of the proof that $\mu(S) \leq 2p$ was wrong. ∎

COROLLARY 3.3. $S(2,q,p) \subseteq DS(2,q,p)$.

PROOF. Let L, l, Q, r, R be a sweep of length 2 of a $S(2)$-tree T having order q and rank p, and consider the vertex $S = [L, l, Q, r, R]$. According to Definition 2.5, the sweep is incompressible; that is, none of S's 2-partitions are proper.

As the order of T is q, the number of keys in each of the three nodes L, Q and R is at least q. Consequently, $|k(S)| > 2q$. Using Lemma 3.2, we have

$$\mu(S) = \mu(L) + \mu(l) + \mu(Q) + \mu(r) + \mu(R) > 2p.$$

This means that any incompressible sweep is dense, which implies that the tree T is a $DS(2)$-tree of order q and rank p. ∎

The following example shows that density of a length 2 sweep is not a sufficient condition for its incompressibility.

EXAMPLE 3.2. Let L, l, Q, r, R be a sweep of length 2 of a $S(2)$-tree T of order q and rank p. Let p be an even natural number divisible by $2q+1$, and

$$\mu(L) = \mu(R) = \frac{p}{2},$$
$$\mu(l) = \mu(r) = 1,$$
$$Q = \langle Q_0, k_1, Q_1, \ldots, k_{q+1}, \ldots, Q_{2q}, k_{2q+1}, Q_{2q+1}\rangle,$$
$$\mu(k_i) = \frac{p}{(2q+1)}, \qquad i = 1, \ldots, 2q+1.$$

Then

$$\mu(L) + \mu(l) + \mu(Q) + \mu(r) + \mu(R) = 2p + 2 > 2p.$$

That is, the sweep L, l, Q, r, R is dense. Consider the following partition of the vertex $[L, l, Q, r, R]$:

$$L' = [L, l, \langle Q_0, k_1, Q_1, \ldots, k_q, Q_q \rangle],$$
$$R' = [\langle Q_{q+1}, k_{q+2}, \ldots, k_{2q+1}, Q_{2q+1} \rangle, r, R].$$

It is easy to see that the partition is proper.

$$[L', k_{q+1}, R'] = [L, l, Q, r, R],$$
$$\mu(L') = \mu(R') = \frac{p}{2} + 1 + \frac{qp}{2q+1} = \frac{p}{2} + 1 + \frac{p}{2} - \frac{p}{2(2q+1)} \leq p,$$
$$|k(L')| > |k(L)| \geq q, \qquad |k(R')| > |k(R)| \geq q.$$

Using Corollary 3.3 and Example 3, we can formulate the following proposition.

PROPOSITION 3.4. *$S(2, q, p) \subset DS(2, q, p)$.*

4. SPACE LOWER BOUNDS FOR $S(2)$-TREES

In this section, we analyze the space efficiency of $DS(2)$-trees. According to Proposition 3.4, this analysis is also valid for $S(2)$-trees.

Contrary to B-trees, we define *utilization* for $DS(b)$-trees as the ratio of the total weight $M(T)$ of a given tree T to the maximal possible weight np of an n-vertex $DS(b)$-tree of rank p

$$\Delta(T) = \frac{M(T)}{np}.$$

Note that the utilization for $DS(b)$-trees is a function of the weight of the tree, rather than of the number of its keys as in the case of B-trees.

4.1. Simple Bounds

Let us begin from simple bounds.

PROPOSITION 4.1. *Let $T \in DS(2, q, p)$ be an n-vertex tree. Then*

$$\Delta(T) \geq \frac{q}{p}.$$

PROOF. Each vertex of the tree contains at least q keys. The weight of each key is at least one. Therefore, for any vertex S, its weight $\mu(S) \geq q$. So the total weight of T is $M(S) \geq qn$, and its utilization is $\Delta(T) = M(T)/np \geq q/p$. ∎

PROPOSITION 4.2. *Let T be an n-vertex $DS(2)$-tree of rank p. If $n \geq 4$, then*

$$\Delta(T) > \frac{4}{9} - \frac{2}{n}.$$

PROOF. Let L_i, l_i, S_i, r_i, R_i, $i = 1, \ldots, s$ be a sequence of sweeps, such that the vertex sets $\{L_i, S_i, R_i\}$ form a disjoint partitioning of the set of the leaves of T. Then

$$M(T) \geq \sum_{i=1}^{s} (\mu(L_i) + \mu(l_i) + \mu(S_i) + \mu(r_i) + \mu(R_i))$$
$$> 2ps.$$

The out-degree of any nonroot internal node of a $DS(2)$-tree is at least 3. And the out-degree of the root is at least 2. It is a general fact for this kind of tree that the number of the tree leaves x is

$$x \geq \frac{2n}{3}.$$

On the other hand, since the sets $\{L_i, S_i, R_i\}$ of neighboring nodes are disjoint, then the number of these sets is

$$s = \left\lfloor \frac{x}{3} \right\rfloor.$$

From this, by definition of the integral part of a number, and using the structure of the trees, it can be derived that $s > 2n/9 - 1$ and hence, $M(T) > 4np/9 - 2p$. That is,

$$\Delta(T) = \frac{M(T)}{np} > \frac{4}{9} - \frac{2}{n}. \qquad \blacksquare$$

This is also a simple lower bound. It is the best we can get in the general case, but as we will see below, utilization as low as this bound is attainable only for a very special case of $DS(2)$-trees. For most cases, we can get a better lower bound. To show that we will use another proof method.

4.2. Height 1 Trees

Let us analyze first the case of height 1 $DS(2)$-trees. We begin with important examples.

EXAMPLE 4.1. Let $T = \langle T_0, k_1, T_1, k_2, T_2 \rangle$ be a $DS(2)$-tree consisting of $n = 4$ vertices. Then by Definition 2.4, $M(t) = \mu(T_0) + \mu(k_1) + \mu(T_1) + \mu(k_2) + \mu(T_2) > 2p$, and hence, $\Delta(T) = M(t)/4p > 1/2$.

EXAMPLE 4.2. Let a tree

$$T = \langle T_0, k_1, T_1, k_2, T_2, k_3, T_3 \rangle$$

have $n = 5$ vertices. Let their weights be $\mu(k_1) = \mu(k_2) = \mu(k_3) = 1$; i.e., $\mu(t) = 3$, and let $\mu(T_0) = \mu(T_3) = 2$, and $\mu(T_1) = \mu(T_2) = p - 1$. This tree is a $DS(2)$-tree and we can see that

$$\begin{aligned} \Delta(T) &= \frac{\mu(t) + \mu(T_0) + \mu(T_1) + \mu(T_2) + \mu(T_3)}{5p} \\ &= \frac{2}{5} + \frac{1}{p}. \end{aligned}$$

For large enough values of p ($p > 10$), the following inequality holds:

$$\frac{2}{5} < \Delta(T) < \frac{1}{2}.$$

Example 4.2 is fully agreed with Proposition 4.2. It shows that the lower bound given by the proposition is almost exact. But it does not mean that we can not get better lower bounds. Later on we are going to show that high branching $DS(2)$-trees utilize space much better.

LEMMA 4.3. *Let $d \geq 3$ be an out-degree of the root node of a rank p $DS(2)$-tree T of height 1.*

(1) *If $d = 0 \bmod 3$, then* $\Delta(T) \geq \dfrac{1}{3}\dfrac{2d}{d+1} + \dfrac{1}{p}\dfrac{d-1}{2(d+1)}$.

(2) *If $d = 1 \bmod 3$, then* $\Delta(T) \geq \dfrac{1}{3}\dfrac{2d-2}{d+1} + \dfrac{1}{p}\dfrac{d-1}{2(d+1)}$.

(3) *If $d = 2 \bmod 3$, then* $\Delta(T) \geq \dfrac{1}{3}\dfrac{2d-1}{d+1} + \dfrac{1}{p}\dfrac{d-1}{2(d+1)}$.

PROOF. Fix $d \geq 3$. Then T has exactly $m = d - 1$ keys in its root node.

Consider a rank p tree $S = \langle S_0, l_1, S_1, \ldots, l_m, S_m \rangle$ of height 1 with the following weights of its vertices:

- $\mu(l_i) = 1/2$ for all $i = 1, \ldots, m$; i.e., the weight of the root is $\mu(S) = m/2$,
- $\mu(S_{3j}) = 0$,
- $\mu(S_{3j+1}) = \mu(S_{3j+2}) = p$, where $(j = 0, \ldots, \lfloor m/3 \rfloor)$.

Note that S is not a structured tree, since the descendants S_i of the root indexed with i divisible by 3 are empty trees, while the others are not. So the root S is neither a leaf, nor an internal vertex. This contradicts Definition 2.2. Besides, we have to accept here unnatural key weights.

By simple counting we have

$$M(S) = \begin{cases} \dfrac{d}{3}2p + \dfrac{m}{2} & \text{if } d = 0 \bmod 3, \\ \dfrac{d-1}{3}2p + \dfrac{m}{2} & \text{if } d = 1 \bmod 3, \\ \dfrac{d-2}{3}2p + p + \dfrac{m}{2} & \text{if } d = 2 \bmod 3. \end{cases}$$

It is clear that S is a b-locally dense tree, and its utilization $\Delta(S) = \frac{M(S)}{(d+1)p}$ for different d's equals the respective right parts of the equations from the hypothesis of the lemma.

We are going to prove that for any $DS(2)$-tree $T = \langle T_0, k_1, T_1, \ldots, k_m, T_m \rangle$ of rank p and of height 1, its weight $M(t)$ is greater or equal to $M(S)$.

From the set $V = \{T_0, T_1, \ldots, T_m, T_{m+1}\}$ of all the vertices of T, where $T_{m+1} = T$, we choose the subset W consisting of those vertices T_i whose weights $\mu(T_i)$ are less than the weights $\mu(S_i)$ of the corresponding vertices of tree S:

$$W = \{T_i \in V \mid \mu(T_i) < \mu(S_i)\}.$$

If $W = \emptyset$, then $M(t) \geq M(S)$. Let $W \neq \emptyset$, then the following two properties hold:

(1) $T \notin W$,

(2) if $T_i \in W$, then either $i = 1 \bmod 3$ or $i = 2 \bmod 3$.

Let $W' = \{T_{i_1}, \ldots, T_{i_r}\}$ be a set of vertices such that for any $j = 1, \ldots, r$, the index $0 < i_j \leq m$ and $i_j = 1 \bmod 3$, and either $T_{i_j} \in W$ or $T_{i_j+1} \in W$.

Consider the sum $M(t)$ and group addends in the following way:

$$\begin{aligned} M(t) = {} & [\mu(T_{i_1-1}) + \mu(k_{i_1}) + \mu(T_{i_1}) + \mu(k_{i_1+1}) + \mu(T_{i_1+1})] + \cdots \\ & + [\mu(T_{i_r-1}) + \mu(k_{i_r}) + \mu(T_{i_r}) + \mu(k_{i_r+1}) + \mu(T_{i_r+1})] + \Sigma_0(T). \end{aligned}$$

Combine addends of the sum $M(S)$ similarly

$$\begin{aligned} M(S) = {} & [\mu(S_{i_1-1}) + \mu(l_{i_1}) + \mu(S_{i_1}) + \mu(l_{i_1+1}) + \mu(S_{i_1+1})] + \cdots \\ & + [\mu(S_{i_r-1}) + \mu(l_{i_r}) + \mu(S_{i_r}) + \mu(l_{i_r+1}) + \mu(S_{i_r+1})] + \Sigma_0(S). \end{aligned}$$

Let us compare the two sums. First, consider $\Sigma_0(T)$, which is the weight sum of the vertices that do not belong to W, plus weights of a number of keys from the root of T. By the definition of set W, and since keys from the root of T cannot have weights less than the corresponding keys from the root of S, we obtain

$$\Sigma_0(T) \geq \Sigma_0(S).$$

Second, for all $j = 1, \ldots, r$,

$$\begin{aligned} \mu(T_{i_j-1}) + \mu(k_{i_j}) + \mu(T_{i_j}) + \mu(k_{i_j+1}) + \mu(T_{i_j+1}) & \\ \geq 2p + 1 = 0 + \frac{1}{2} + p + \frac{1}{2} + p & \\ = \mu(S_{i_j-1}) + \mu(l_{i_j}) + \mu(S_{i_j}) + \mu(l_{i_j+1}) + \mu(S_{i_j+1}). & \end{aligned}$$

It means that $M(T) \geq M(S)$, and hence $\Delta(T) \geq \Delta(S)$. ∎

COROLLARY 4.4. *Let $d \geq 3$ be an out-degree of the root node of a rank p $DS(2)$-tree T of height 1. Then:*

(1) *If $d = 4$, then $\Delta(T) \geq \frac{2}{5} + \frac{3}{10p}$.*

(2) *If $d \neq 4$, then $\Delta(T) \geq \frac{1}{2} + \frac{1}{4p}$.*

(3) *For all d, $\Delta(T) \geq \left(\frac{2}{3} + \frac{1}{2p}\right)\frac{d-1}{d+1}$.*

4.3. The Main Theorem

The following simple combinatorial inequality will be actively used below.

LEMMA 4.5. *Let $\varepsilon, a_i, b_i (i = 1, \ldots, m)$ be positive real numbers such that $a_i/b_i \geq \varepsilon$. Then*

$$\frac{\sum_{i=1}^{m} a_i}{\sum_{i=1}^{m} b_i} \geq \varepsilon.$$

THEOREM 4.6. *Let $T \in DS(2, q, p)$ be an n-vertex tree.*

(1) *In the general case, for $n > \frac{4p}{3}$, $\Delta(T) > \frac{2}{5}$.*

(2) *If none of the internal vertices of T has out-degree 4, then for $n > 2p$, $\Delta(T) > \frac{1}{2}$.*

(3) *If $q \geq 6$, then for $n > \frac{4(q+1)p}{3}$, $\Delta(T) > \frac{2}{3} - \frac{4}{3(q+2)}$.*

PROOF. Consider the following partition of the vertex set V of T. First choose subsets V_i $(i = 1, \ldots, s_1)$. Each V_i consists of all leaves of T outgoing from the common ancestor and of the ancestor itself. Consider now a tree T' obtained from the initial tree T by discarding the vertices that belong to the union of the sets V_i. Let us consider a set of leaves of T' and their predecessors, and let us construct new subsets V_{s_1+i} $(i = 1, \ldots, s_2)$ from them in the same manner as it was done for tree T. Throw out the selected vertices from T' and continue the process until either the remaining tree is empty or it consists of the unique root vertex T. Let the subsets $V_1, \ldots, V_s$ have been built to that moment. In the former case, when the empty tree was obtained, let $V_{s+1} = \emptyset$, and in the latter case let $V_{s+1} = \{T\}$. Therefore, we got a partition $V = V_1 \cup \cdots \cup V_s \cup V_{s+1}$ such that $V_i \cap V_j = \emptyset$ for $i \neq j$. Note, that $V_{s+1} = \emptyset$ iff the height of the tree is odd.

Each vertex set V_i $(i \leq s)$ together with the edges that connect them in T determines a $DS(2)$-tree of height 1. For referring to these trees we will use the same symbols V_i as for their vertex sets. Let $n_i = |V_i|$. Then $\sum_{i=1}^{s+1} n_i = n$ and also $\sum_{i=1}^{s+1} M(V_i) = M(T)$.

Let us prove Estimate 3. Let $a = (d-1)/(d+1)$, where $d = q + 1 \geq 7$ is a lower bound for the out-degrees of the internal vertices of T. Using Estimate 3 from Corollary 4.4, we conclude that the utilization of a height 1 $DS(2)$-tree is evaluated from above by application of a monotonically increasing function of the out-degree of the tree root. Therefore, for all $i \leq s$ and for $\varepsilon = a(2/3 + 1/(2p))$ we have

$$\frac{M(V_i)}{n_i p} \geq \varepsilon.$$

Consider the case when $V_{s+1} \neq \emptyset$, that is, the height of T is even. Then

$$\Delta(T) = \frac{M(T)}{np} = \frac{\sum_{i=1}^{s} M(V_i) + M(T)}{np} \geq \frac{\sum_{i=1}^{s} M(V_i) + 1}{np}$$

$$= \frac{\sum_{i=1}^{s} M(V_i) + (2pa/3+1) - 2pa/3}{np}$$
$$= \frac{\sum_{i=1}^{s} M(V_i) + (2pa/3+1)}{\sum_{i=1}^{s} n_i p + p} - \frac{2a}{3n}.$$

Since $\frac{2pa/3+1}{p} > \varepsilon$ and $\frac{M(V_i)}{n_i p} \geq \varepsilon$ $(i \leq s)$, using Lemma 4.5 we get Estimate 3 of the theorem. For $n > 4p/3$,

$$\Delta(T) \geq \frac{2a}{3} + \frac{a}{2p} - \frac{2a}{3n} > \frac{2}{3}a = \frac{2}{3}\frac{q}{q+2} = \frac{2}{3} - \frac{4}{3(q+2)}.$$

Now consider the case when $V_{s+1} = \emptyset$, that is, the height of T is odd.

If the out-degree of the root node of T is not less than d, then by Lemma 4.5 and Corollary 4.4, we easily get

$$\Delta(T) = \frac{M(T)}{np} = \frac{\sum_{i=1}^{s} M(V_i)}{\sum_{i=1}^{s} n_i p} \geq \varepsilon > \frac{2}{3}a.$$

If the out-degree of the root node of $T = \langle T_0, k_1, T_1, \ldots, k_m, T_m \rangle$ is less than d, that is, $m < q$, then by Estimate 3 of Corollary 4.4,

$$\frac{M(V_i)}{n_i p} \geq \varepsilon$$

for all $(i < s)$, and by Lemma 4.1 we have

$$\frac{M(V_s)}{n_s p} > \frac{1}{p}$$

where $n_s = m + 2$.

Utilization in this case is

$$\Delta(T) = \frac{M(T)}{np} \geq \frac{\sum_{i=1}^{s-1} M(V_i) + M(V_s)}{np}$$
$$= \frac{\sum_{i=1}^{s-1} M(V_i) + (2n_s pa/3 + M(V_s)) - 2n_s pa/3}{np}$$
$$= \frac{\sum_{i=1}^{s-1} M(V_i) + (2n_s pa/3 + M(V_s))}{\sum_{i=1}^{s-1} n_i p + n_s p} - \frac{2n_s a}{3n}.$$

We see that

$$\frac{2n_s pa/3 + M(V_s)}{n_s p} = \frac{2a}{3} + \frac{M(V_s)}{n_s p} > \frac{2a}{3} + \frac{1}{p} > \varepsilon$$

since $a/(2p) < 1/(2p) < 1/p$.

Now using Lemma 4.5, and the fact that $n_s = m + 2 \leq d$, for all $n > 4dp/3$ we obtain

$$\Delta(T) \geq \varepsilon - \frac{2n_s a}{3n} \geq \frac{2a}{3} + \frac{a}{2p} - \frac{2da}{3n} > \frac{2}{3}a.$$

This completes the proof of Estimate 3.

Similarly, using Lemma 4.5 and Corollary 4.4, one can prove Estimates 1 and 2. ∎

Analyzing the above results, I would like to conclude that in the general case (Estimate 1), when the out-degrees of internal nodes are not bounded, we have 2/5 lower bound, which is close to the bound given by Proposition 4.2. And as it can be seen from Example 4.2, this bound is almost exact. But the trees with that low utilization have very specific structure. Namely, almost all of their internal nodes should have out-degree 4. Otherwise the lower bound rises up to 1/2 (Estimate 2). And, finally, Estimate 3 guarantees that the higher the branching of the tree is, the higher lower bound of utilization it has. The limit of these lower bounds is 2/3, which is the upper bound of the lower bounds for utilization of $DS(2)$-trees.

5. ALGORITHMS

In this section, an informal description of algorithms for search, insertion and deletion operations for the class $S(2,q,p)$ is presented.

5.1. Search

The *search* operation for $S(2)$-trees is performed in the usual way for the tree data structure. The search algorithm for a key k in a $S(2)$-tree T consists of at most logarithmically many steps of local searches in the tree nodes.

We will refer to this algorithm as the procedure SEARCH(T,k). The result of the procedure is a three-tuple $(\mathit{TrueValue}, S, k_i)$. *TrueValue* determines whether the key k is found in T. S is a vertex that was examined last by the procedure, and k_i is the minimal key from S that is greater than or equal to k. If all keys in S are less than k, then SEARCH returns a special value which is denoted by k_{m+1}, where $m = |k(S)|$.

5.2. Insertion

Informally, the *insertion* algorithm is as follows. First it verifies by using the procedure SEARCH(T,k) whether the given key k is already contained in the $S(2)$-tree T. If it is the case, then insertion is finished. If not, then procedure SEARCH returns a leaf $S = \langle S_0, k_1, S_1, \ldots, k_m, S_m\rangle$ and a key k_i in it, before which the key k must be inserted. Next, k is inserted to the given place of the vertex S. And the insertion procedure proceeds to balancing the tree. Balancing is performed by the procedure-function BALANCE, which is the main common part of both the insertion and the deletion algorithms. BALANCE starts from the leaf S and balances all the nodes that lay on the path from the root T to the enlarged leaf S.

5.3. Deletion

Similarly to the insertion, the *deletion* procedure begins with search. Given a key k and a $S(2)$-tree T, the procedure SEARCH looks for k in T. If T does not contain k, then the deletion is finished. Otherwise SEARCH returns a vertex $S = \langle S_0, k_1, S_1, \ldots, k_m, S_m\rangle$ and a key $k_i = k$ in it, that must be deleted.

The case when S is not a leaf can easily be reduced to the case of deletion from a leaf. Indeed, if S is not a leaf, then $S_i \neq \Lambda$. Let's replace the key k_i in S by the minimal (according to the order $\ll$ on K) key k' from the set $K(S_i)$. It is clear that k' belongs to some leaf S', which is the leftmost node of the subtree S. If the replacement of the key k_i in S by k' breaks the balance conditions for S, then they will be restored while balancing the tree.

Now let S be a leaf. The deletion procedure removes the given key from S and proceeds to balance the tree. Balancing is again performed by procedure BALANCE started at leaf node S.

5.4. Balancing

The main part of the insertion and deletion algorithms is the procedure-function BALANCE. The balancing begins from the leaf node S, which is given as an input parameter for the procedure. After working on the level of the current node S, the procedure takes for balancing the direct ancestor of S. The process proceeds further up to the root node. The balanced tree is returned as the result of procedure BALANCE.

For any current vertex S, the procedure decides to balance S if one of the following three conditions holds.

(1) The weight of S is small, that is, one of the three sweeps of length 3, containing S, is not incompressible.
(2) $|k(S)| < q$.
(3) $\mu(S) > p$.

If the cause of broken balance of the current node S is Condition 1, then procedure BALANCE_B is used for balancing S. If Condition 2 holds for S, then BALANCE_C is used. And in the case of Condition 3, BALANCE_W is used. When the conditions don't hold, BALANCE skips the level.

Each of the three procedures restores the structure of $S(2)$-tree disturbed locally for one or two vertices of the current tree level. While correcting the structure of the tree on the current level, the algorithm should also change the ancestors of S. This can break in turn the balance conditions for the lower level nodes. Such breakdowns are also local, since no more than two lower level nodes can be changed. Each of these nodes is either the direct ancestor of S, or one of its neighbors. Coming to the next level of the tree procedure, BALANCE merges the two renewed nodes of the level, and balances them as a whole.

Computation is stopped after coming through the tree root. Thus, the algorithm examines all the nodes that lay on the path from the tree root to the leaf, containing the new key, and balances them if necessary. Only these nodes and their neighbors (up to the second from the left and from the right) in the tree can be transformed by the algorithm.

The explicit algorithms are outlined in the Appendix. The time complexity of the algorithms gives the following proposition.

PROPOSITION 5.1. *Search, insertion and deletion of a key in an n-vertex $S(2)$-tree can be performed in time $O(\log n)$.*

APPENDIX

DEFINITIONS

Let T be a structured tree and S be its vertex.

- S^* denotes the direct ancestor of vertex S in T.
- Sweep $\langle L_2, l_2, L_1, l_1, S, r_1, R_1, r_2, R_2\rangle$ of tree T is called a 2-*vicinity* of vertex S. Node F denotes the direct ancestor of S. Node FL_i (FR_i) denotes an ancestor of S that contains delimiting key l_i (r_i). L_i, l_i, R_i, r_i, FL_i, FR_i are (local) variables of the procedures.
- LNF and RNF are global variables of the procedures that are intended to indicate whether the left ($LNF \neq \langle\rangle$) or the right ($RNF \neq \langle\rangle$) neighbors of vertex S^* were changed while balancing of the level of vertex S. Note that at any time of balancing only one of the two variables can be nonempty.
- $WW(S) \rightleftharpoons \mu(S) \leq p$.
- $WC(S) \rightleftharpoons |k(S)| \geq q$.
- $WF(S) \rightleftharpoons WW(S)$ and $WC(S)$.
- $IC(A, a, B, b, C)$ means by definition that sweep A, a, B, b, C is incompressible.
- $WBL(S) \rightleftharpoons IC(L_2, l_2, L_1, l_1, S)$.
- $WBC(S) \rightleftharpoons IC(L_1, l_1, S, r_1, R_1)$.
- $WBR(S) \rightleftharpoons IC(S, r_1, R_1, r_2, R_2)$.
- $WB(S) \rightleftharpoons WBL(S)$ and $WBC(S)$ and $WBR(S)$.
- $W2N(S) \rightleftharpoons WF(S)$ and $WB(S)$.

INSTRUMENTAL PROCEDURES

The following procedures and functions are used for describing the algorithms.

- Procedure *MakeCurrent*(S) initializes local variables L_i, l_i, R_i, r_i, FL_i, FR_i according to S.
- Procedure *Replace*(S, P, Q) replaces a part of vertex S (of the tree), that coincides with P, with Q.
- Functions *LeftOf*(S, k) and *RightOf*(S, k) define two parts of vertex S, that are on the left and on the right, respectively, of key k.

- Three functions

$$\begin{aligned} &\textit{Moveleft} && (A,a,B,b,C) \longrightarrow (P,d,Q)\\ &\textit{MoveRight} && (A,a,B,b,C) \longrightarrow (P,d,Q)\\ &\textit{Split} && (A,a,B,b,C) \longrightarrow (P,d,Q) \end{aligned}$$

are applied to the not incompressible sweep A,a,B,b,C, and result P,d,Q is a proper 2-partition of vertex $[A,a,B,b,C]$. Additional input and output conditions for the functions are given by the following table.

Function	Input	Output
Moveleft	$WF(A)$ and $WF(B)$	$k(A) \subseteq k(P)$
MoveRight	$WF(B)$ and $WF(C)$	$k(C) \subseteq k(Q)$
Split	$WF(A)$ and $WF(C)$	$k(A) \subseteq k(P)$
		$k(C) \subseteq k(Q)$

- In BALANCE_W we use procedure *ComputeSets*(S), which computes seven subsets of $k(S)$. The subsets control the computational process of BALANCE_W and are defined by

$$\begin{aligned} \textit{MLeft} &= \{d \in k(S) \mid IC\,(L_2,l_2,L_1,l_1,\textit{LeftOf}(S,d))\}\,,\\ \textit{MRight} &= \{d \in k(S) \mid IC\,(\textit{RightOf}(S,d),r_1,R_1,r_2,R_2)\}\,,\\ \textit{MDelimL} &= \{d \in k(S) \mid WF\,(\textit{RightOf}(S,d))\}\,,\\ \textit{MDelimR} &= \{d \in k(S) \mid WF\,(\textit{LeftOf}(S,d))\}\,,\\ \textit{MDelim} &= \textit{MDelimL} \cap \textit{MDelimR},\\ \overline{\textit{MLeft}} &= k(S) \setminus \textit{MLeft},\\ \overline{\textit{MRight}} &= k(S) \setminus \textit{MRight}. \end{aligned}$$

THE MAIN PROCEDURES

Procedure INSERTION(T,k)
$(\textit{TrueValue}, S, k_i) :=$ SEARCH(T,k);
/*** $S = \langle \Lambda, k_1, \Lambda, \ldots, k_{i-1}, \Lambda, k_i, \Lambda, \ldots, k_m, \Lambda\rangle$ ***/
if *TrueValue* $=$ *TRUE* **then stop**; **fi**
$\textit{Replace}(S, \langle k_i\rangle, \langle k, \Lambda, k_i\rangle)$;
/*** $S = \langle \Lambda, k_1, \Lambda, \ldots, k_{i-1}, \Lambda, k, \Lambda, k_i, \Lambda, \ldots, k_m, \Lambda\rangle$ ***/
$T :=$ BALANCE(S);
stop;
End_of_Procedure

Procedure DELETION(T,k)
$(\textit{TrueValue}, S, k_i) :=$ SEARCH(T,k);
/*** $S = \langle S_0, k_1, S_1, \ldots, S_{i-1}, k_i, S_i, \ldots, k_m, S_m\rangle$ ***/
if $TrueValue = FALSE$ **then stop**; **fi**
if $S_0 \neq \Lambda$ **then** /* S is not a leaf */
  $(S', k') := \textit{MinKey}(S_i)$;
  $\textit{Replace}(S, \langle k\rangle, \langle k'\rangle)$;
  $S = S'$;
  $k = k'$;
fi
/*** $S = \langle \Lambda, k_1, \Lambda, \ldots, k_{i-1}, \Lambda, k, \Lambda, k_i, \Lambda, \ldots, k_m, \Lambda\rangle$ ***/
$\textit{Replace}(S, \langle \Lambda, k, \Lambda\rangle, \langle \Lambda\rangle)$;
$T :=$ BALANCE(S);
stop;
End_of_Procedure

```
Procedure BALANCE(S)
LNF := RNF := ⟨⟩;
while S ≠ ⟨⟩  do
      X := S;
A:    if      ¬WB(S)  then  P := BALANCE_B(S);
      else if ¬WC(S)  then  P := BALANCE_C(S);
      else if ¬WW(S)  then  P := BALANCE_W(S);
      else                  P := S*;
      fi fi fi
B:    if LNF ≠ ⟨⟩ then
        P := MakeCurrent(P);
        Q := [L1, l1, P];
        if FL1 = F then
          Replace(F, ⟨L1, l1, P⟩, ⟨Q⟩);
          LNF := ⟨⟩;
        else
          Replace(F, ⟨P⟩, ⟨Q⟩);
          Replace(FL1, ⟨l1⟩, ⟨l2⟩);
          Replace(FL2, ⟨L2, l2, L1⟩, ⟨L2⟩);
          LNF := FL2;
        fi
        S := Q;
C:    else if RNF ≠ ⟨⟩ then
        P := MakeCurrent(P);
        Q := [P, r1, R1];
        if F = FR1 then
          Replace(F, ⟨P, r1, R1⟩, ⟨Q⟩);
          RNF := ⟨⟩;
        else
          Replace(F, ⟨P⟩, ⟨Q⟩);
          Replace(FR1, ⟨r1⟩, ⟨r2⟩);
          Replace(FR2, ⟨R1, r2, R2⟩, ⟨R2⟩);
          RNF := FR2;
        fi
        S := Q;
      else S := P;
      fi fi
od
return(X);
End_of_Procedure

Procedure BALANCE_B(S)
S := MakeCurrent(S);
BA:  if ¬IC([L2, l2, L1, l1, S]) then
       (Q2, l, Q1) := Moveleft(L2, l2, L1, l1, S);
BA1:  if FL2 = FL1 = F then
        Replace(F, ⟨L2, l2, L1, l1, S⟩, ⟨Q2, l, Q1⟩);
BA2:  else if FL1 = F then    /* FL2 ≠ F */
        Replace(F, ⟨L1, l1, S⟩, ⟨Q1⟩);
        Replace(FL2, ⟨l2⟩, ⟨l⟩);
        Replace(L2, L2, Q2);
BA3:  else    /* FL2 ≠ F  &  FL1 ≠ F */
        Replace(F, ⟨S⟩, ⟨Q1⟩);
        Replace(FL1, ⟨l1⟩, ⟨l⟩);
        Replace(FL2, ⟨L2, l2, L1⟩, ⟨Q2⟩);
```

```
        LNF := FL2;
      fi fi
      S := MakeCurrent(Q1);
    fi
BB: if ¬IC([S, r1, R1, r2, R2]) then
      (P1, r, P2) := MoveRight(S, r1, R1, r2, R2);
BB1:  if F = FR1 = FR2 then
        Replace(F, ⟨S, r1, R1, r2, R2⟩, ⟨P1, r, P2⟩);
BB2:  else if F = FR1 then    /* F ≠ FR2 */
        Replace(F, ⟨S, r1, R1⟩, ⟨P1⟩);
        Replace(FR2, ⟨r2⟩, ⟨r⟩);
        Replace(R2, R2, P2);
BB3:  else    /* F ≠ FR1 & F ≠ FR2 */
        Replace(F, ⟨S⟩, ⟨P1⟩);
        Replace(FR1, ⟨r1⟩, ⟨r⟩);
        Replace(FR2, ⟨R1, r2, R2⟩, ⟨R2⟩);
        RNF := FR2;
      fi fi
      S := MakeCurrent(P1);
    fi
BC: if ¬IC([L1, l1, S, r1, R1]) then
      (Q, d, P) := Split(L1, l1, S, r1, R1);
BC1:  if FL1 = F = FR1 then
        Replace(F, ⟨L1, l1, S, r1, R1⟩, ⟨Q, d, P⟩);
BC2:  else if FL1 ≠ F & F = FR1 then
        Replace(F, ⟨S, r1, R1⟩, ⟨P⟩);
        Replace(FL1, ⟨l1⟩, ⟨d⟩);
        Replace(L1, L1, Q);
BC3:  else if FL1 = F & F ≠ FR1 then
        Replace(F, ⟨L1, l1, S⟩, ⟨Q⟩);
        Replace(FR1, ⟨r1⟩, ⟨d⟩);
        Replace(R1, R1, P);
BC4:  else    /* FL1 ≠ F & F ≠ FR1 & F = ⟨S⟩ */
        if LNF = FL2 then
          Replace(F, ⟨S⟩, ⟨Q⟩);
          Replace(FR1, ⟨r1⟩, ⟨d⟩);
          Replace(R1, R1, P);
          Replace(FL1, ⟨l1⟩, ⟨l2⟩);
          Replace(FL2, ⟨L2, l2, L1⟩, ⟨L2⟩);
        else    /* LNF = ⟨⟩ */
          Replace(F, ⟨S⟩, ⟨P⟩);
          Replace(FL1, ⟨l1⟩, ⟨d⟩);
          Replace(L1, L1, Q);
          Replace(FR1, ⟨r1⟩, ⟨r2⟩);
          Replace(FR2, ⟨R1, r2, R2⟩, ⟨R2⟩;
          RNF := FR2;
        fi
      fi fi fi
    fi
return(F);
End_of_Procedure

Procedure BALANCE_C(S)
S := MakeCurrent(S);
if F = ⟨⟩ then
```

```
  if S = ⟨S_0⟩ then
    Replace(S, S, ⟨⟩);
    return(S_0);
  fi
  return(F);
fi
CA: if |k(F)| > 1 then
      if FL_1 = F then
        Q := [L_1, l_1, S];
        Replace(F, ⟨L_1, l_1, S⟩, ⟨Q⟩);
      else   /* F = FR_1 */
        Q := [S, r_1, R_1];
        Replace(F, ⟨S, r_1, R_1⟩, ⟨Q⟩);
      fi
CA1:  else   /* |k(F)| = 1 & LNF = RNF = ⟨⟩ */
      if FL_1 ≠ F then
        Q := [L_1, l_1, S];
        Replace(F, ⟨S⟩, ⟨Q⟩);
        Replace(FL_1, ⟨l_1⟩, ⟨l_2⟩);
        Replace(FL_2, ⟨L_2, l_1, L_1⟩, ⟨L_2⟩);
        LNF := FL_2;
      else   /* F ≠ FR_1 */
        Q := [S, r_1, R_1];
        Replace(F, ⟨S⟩, ⟨Q⟩);
        Replace(FR_1, ⟨r_1⟩, ⟨r_2⟩);
        Replace(FR_2, ⟨R_1, r_2, R_2⟩, ⟨R_2⟩);
        RNF := FR_2;
      fi
    fi
CB: if μ(Q) > p then
      S := MakeCurrent(S);
      F := BALANCE_W(S);
    fi
return(F);
End_of_Procedure

Procedure BALANCE_W(S)
S:= MakeCurrent(S);
ComputeSets(S);
WA: if MLeft ∩ MRight ≠ ∅ then
      /** choose element from the intersection **/
      d ∈ MLeft ∩ MRight;
      (Q_2, l, Q_1) := Moveleft(L_2, l_2, L_1, l_1, LeftOf(S, d));
      (P_1, r, P_2) := MoveRight(RightOf(S, d), r_1, R_1, r_2, R_2);
      Replace(L_2, L_2, Q_2);
      Replace(FL_2, ⟨l_2⟩, ⟨l⟩);
      Replace(R_2, R_2, P_2);
      Replace(FR_2, ⟨r_2⟩, ⟨r⟩);
WA1:  if FL_1 = F = FR_1 then
        Replace(F, ⟨L1, l_1, S, r_1, R_1⟩, ⟨Q_1, d, P_1⟩);
WA2:  else if FL_1 ≠ F then   /* F = FR_1 */
        Replace(F, ⟨S, r_1, R_1⟩, ⟨P_1⟩);
        Replace(FL_1, ⟨l_1⟩, ⟨d⟩);
        Replace(L_1, L_1, Q_1);
        LNF := FL_2;
```

```
WA3:  else   /* FL_1 = F  &  F ≠ FR_1 */
        Replace(F, ⟨L_1, l_1, S⟩, ⟨Q_1⟩);
        Replace(FR_1, ⟨r_1⟩, ⟨d⟩);
        Replace(R_1, R_1, P_1);
        RNF := FR_2;
      fi  fi
      return(F);
    fi
WB: if MLeft ≠ ∅ then
      if MDelimL ∩ MLeft ≠ ∅ then d ∈ MDelimL ∩ MLeft;
      else                                d := max(MLeft);   fi
      (Q_2, l, Q_1) := Moveleft(L_2, l_2, L_1, l_1, LeftOf(S, d));
      P := RightOf(S, d);
      Replace(F, ⟨S⟩, ⟨P⟩);
      Replace(FL_1, ⟨l_1⟩, ⟨d⟩);
      Replace(L_1, L_1, Q_1);
      Replace(FL_2, ⟨l_2⟩, ⟨l⟩);
      Replace(L_2, L_2, Q_2);
      if FL_1 ≠ F  then LNF := FL_2;  fi
      if WW(P)  then  return(F);  fi
      S := MakeCurrent(P);
      ComputeSets(S);
    fi   /* MLeft = k(S) */
WC: if MRight ≠ ∅ then
      if MDelimR ∩ MRight ≠ ∅ then d ∈ MDelimR ∩ MRight;
      else                                d := min(MRight);   fi
      Q := LeftOf(S, d);
      (P_1, r, P_2) := MoveRight(RightOf(S, d), r_1, R_1, r_2, R_2);
      Replace(F, ⟨S⟩, ⟨Q⟩);
      Replace(FR_1, ⟨r_1⟩, ⟨d⟩);
      Replace(R_1, R_1, P_1);
      Replace(FR_2, ⟨r_2⟩, ⟨r⟩);
      Replace(R_2, R_2, P_2);
      if F ≠ FR_1  then RNF := FR_2;  fi
      if WW(Q)  then  return(F);  fi
      S := MakeCurrent(Q);
      ComputeSets(S);
    fi   /* MRight = k(S) */
WD: if MDelim ≠ ∅ then
      d ∈ MDelim;
      Q := LeftOf(S, d);
      P := RightOf(S, d);
      Replace(F, ⟨S⟩, ⟨Q, d, P⟩);
      return(F);
    fi
WE: /* MLeft = MRight = k(S)  &  MDelim = ∅ */
    d := max(MDelimR);
    Q := LeftOf(S, d);
    P := RightOf(S, d);
    Replace(F, ⟨S⟩, ⟨Q, d, P⟩);
    F := BALANCE_W(P);
return(F);
End_of_Procedure
```

REFERENCES

1. A.P. Pinchuk and K.V. Shvachko, Maintaining dictionaries: Space-saving modifications of B-trees, In *Lecture Notes in Computer Science*, Vol. 646, (1992).
2. K.V. Shvachko, Generalization of B-trees for variable-length records (to appear).
3. D. Wood, *Data Structure, Algorithms, and Performance*, Addison-Wesley, (1993).
4. G.M. Adel'son-Vel'skii and E.M. Landis, An algorithm for the organization of information, *Soviet Math. Doklady* **3**, 1259–1262 (1972).
5. C.C. Foster, A generalization of AVL-trees, *Communications of the ACM* **16**, 513–517 (1973).
6. H.R. Lewis and L. Denenberg, *Data Structures and Their Algorithms*, Harper-Collins, New York, (1991).
7. L.J. Guibas and R. Sedgewick, A dichromatic framework for balanced trees, In *Proceedings, 19^{th} Annual IEEE Symposium on Foundations of Computer Science*, pp. 8–12, (1978).
8. A.C.-C. Yao, On random 2-3 trees, *Acta Inf.* **9**, 159–170 (1978).
9. R. Bayer, Symmetric binary B-tree: Data structure and maintenance algorithms, *Acta Inf.* **1** (4), 290–306 (1972).
10. R. Bayer and E. McCreight, Organization and maintenance of large ordered indexes, *Acta Inf.* **1** (3), 173–189 (1972).
11. D.E. Knuth, *The Art of Computer Programming*, Vol. 3 (Sorting and Searching), Addison-Wesley, Reading, MA, (1973).
12. N. Wirth, *Algorithms and Data Structure*, Prentice-Hall, Englewood Cliffs, NJ, (1986).
13. T.J. Teorey and D.P. Fry, *Design of Database Structures*, Vol. 2, Prentice-Hall, Englewood Cliffs, NJ, (1982).
14. K.V. Shvachko, Space-saving modifications of B-trees, In *Proceedings of Symposium on Computer Systems and Applied Mathematics*, St. Petersburg, p. 214, (1993).
15. M.M. Gilula, *The Set Model for Database and Information Systems*, Addison-Wesley (in association with ACM Press), Wokingham, (1994).
16. D. Comer, The ubiquitous B-tree, *Comp. Surv.* **11** (2), 121–137 (1979).
17. G.K. Gupta and B. Srinivasan, Approximate storage utilization of B-tree, *Inf. Proc. Lett.* **22**, 243–246 (1986).
18. S.-H.S. Huang, Height-balanced trees of order (β,γ,α), *ACM Trans. Database Syst.* **10** (2), 261–284 (1985).
19. T. Johnson and D. Shasha, Utilization of B-tree with inserts, deletes, and modifies, In *Proceedings of the ACM Principles of Database Systems Symposium*, pp. 235–244, (1989).
20. C.H.C. Leung, Approximate storage utilization of B-tree: A simple derivation and generalizations, *Inf. Proc. Lett.* **19**, 199–201 (1984).
21. E.M. McCreight, Pagination of B^*-trees with variable-length records, *Commun. ACM* **20** (9), 670–674 (1977).
22. A.L. Rosenberg and L. Snyder, Time- and space-optimality in B-tree, *ACM Trans. Database Syst.* **6** (1), 174–193 (1981).
23. W.E. Wright, Some average performance measure for the B-tree, *Acta Inf.* **21**, 541–557 (1985).