S(b)-Trees: An Optimal Balancing of Variable Length Keys

Konstantin V. Shvachko
Fremont, CA, U.S.A.
kshvachko@4ds.com

Abstract

The paper studies the problem of maintaining **external dynamic dictionaries** *with variable length keys. We introduce a new type of* **balanced trees**, *called $S(b)$-trees, which generalize traditional B-***trees***. Contrary to B-trees $S(b)$-trees provide optimal utilization of keys of variable length, while the data access time remains logarithmical, the same as for B-trees. The main property of the new 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 nodes. We prove $1-\varepsilon$ utilization lower bound for these trees where ε is inversely proportional to the tree branching. Logarithmic running time algorithms for search, insertion, and deletion are presented. The data structure is a flexible storage solution for semi-structured data and XML databases.*

Keywords: dynamic dictionaries, balanced trees, B-trees, semi–structured data, XML databases.

1. Introduction

The problem of *maintaining dynamic dictionaries* (MDD) is considered to be a central problem of the theory of data structures. Let a set K of dictionary elements, called *keys*, be given. For any finite subset D of K and for any key k three operations of *search*, *insertion*, and *deletion* are defined as follows

$$\begin{aligned} \textit{Search}(D,k) &= k \in D \\ \textit{Insert}(D,k) &= D \cup \{k\} \\ \textit{Delete}(D,k) &= D \setminus \{k\} \end{aligned}$$

The problem is to provide space–efficient way of storing keys, as well as time–efficient algorithms for performing the operations.

For linearly ordered key sets searching can be performed in logarithmic on the number of keys time. Otherwise, only the exhaustive search algorithm is applicable. It is also proven [20] that searching of a key in a finite linearly ordered set is lower–bounded by logarithm on the number of keys in the set. Therefore, logarithm is the optimum for searching in linearly ordered sets. The same lower bound is true for insertions and deletions, since in order to insert or delete a key it is particularly necessary to check whether the key is contained in the input set.

Balanced trees are considered to be a standard solution for the MDD problem. Historically, first solutions of the problem were based on *binary trees*. Each node of a binary tree contains only one key and has at most two children, such that the key in the left child is less while the key in the right child is greater than the key contained in the given node. Searching in such a tree is logarithmic only if the tree is balanced, otherwise the tree can degenerate into a list structure yielding a linear–time exhaustive search in the worst case.

An absolutely balanced tree [18] is a regular, that is each node out-degree is either 2 (internal node) or 0 (leaf), binary tree such that the length of any path from the root of the tree to a leaf equals either the tree height h or $h-1$. A search in an absolutely balanced tree is obviously logarithmic, since $h = \lceil \log_2 n \rceil$. Unfortunately, in order to support the absolute balance condition while insertions and deletions a substantial reconstruction of the tree may be required such that the algorithms become linear rather than logarithmic in time [18], [19].

So–called *week balance conditions* were developed in order to avoid inefficiency of insertions and deletions while preserving the asymptotic optimality of searching.

The first example of a balanced tree with all the three operations optimal in time was proposed by G.M. Adelson-Velskii and E.M. Landis [1], [4]. An AVL-tree is a regular binary tree satisfying the following week balance condition: for any tree node the difference between the heights of its two child subtrees is at most 1.

Another variant of week balance condition is used in 2-3-trees (J.Hopcroft, 1970, [9]). All leaves 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 [6], [21].

A generalization of 2-3-trees, called B-trees, was proposed by R.Bayer [2], [3]. In B-trees like in 2-3-trees all

paths from the root to the leaves have the same length. The generalization is that the number of keys in a B-tree node is between q and $2q$, for a fixed parameter q, called the tree order.

By now, a number of variants of the classical B-trees have been examined in literature. B^*-trees [8], [18], B^+-trees [20], symmetric B-trees [2], [19], and (a, b)-trees [9] are among them.

For a B-tree T of order q, composed of n nodes, its *utilization* is the fraction $\delta(T) = \frac{\textit{number of keys in } T}{2qn}$. Since the number of keys in an n-node B-tree varies between qn and $2qn$, the lower bound of utilization for B-trees is $1/2$.

The disadvantages of B-trees have been widely discussed [8], [12], [17] in the context of this space lower bound. The bottom line is that in B-trees each key is treated as a unit and the key weight, that is the amount of space required for storing the key, is not taken into account. Hence, B-trees utilize memory well only when the keys have (almost) identical weight, while when the keys differ greatly in weight, they lead to an exhaustive waste of memory. Namely, it is not possible to guarantee any lower bound of utilization for B-trees greater than 0 when key weights are taken into account.

In order to store variable weight keys efficiently in a balanced tree, its node capacity should be determined based on the total weight of keys in a node rather than on the number of keys per node, and the balance conditions should be re-formulated in adequate terms.

Initially the idea was mentioned by D.Knuth [8] (with a reference to an unpublished result of T.H.Martin). It was precisely formulated and analyzed in [12].

E.M. McCreight [10] describes a strategy for allocating variable length keys across nodes of a B^*-tree. The idea is to place smaller keys into higher level internal tree nodes in order to increase their branching. "This strategy results in shallow trees with fast access time".

This paper presents a generalized approach to the solution of the MDD problem with variable weight keys. We introduce a class $S(b, q, p)$ of balanced trees, called $S(b)$-*trees of order* q *and rank* p, which is characterized by the following properties.

1. The total weight of keys contained in one tree node does not exceed p.

2. The number of keys in a non-root node of the tree is at least q.

3. The tree is b-*locally incompressible*, meaning informally that for any $b+1$ neighboring nodes of the same level of the tree the nodes cannot be "compressed" into b nodes satisfying the first two properties.

Incompressibility is a week balance condition for $S(b)$-trees.

Additionally, we introduce a class $DS(b, q, p)$ of balanced trees, called $DS(b)$-*trees* that satisfy properties 1 and 2 above, but have another week balance condition

3'. The tree is b-*locally dense*, meaning that the total weight of keys stored in any $b + 1$ neighboring nodes of the tree together with the delimiting for these nodes keys is greater than bp.

In the paper we show that if all keys are of equal weight 1, then the class of B-trees of order q coincides with the class $S(0, q, 2q)$, and that the class of 2-3-trees is $S(0, 1, 2)$.

Further, we study the relationships between the tree classes. We prove that the classes of $S(b)$-trees and $DS(b)$-trees coincide for $b = 0, 1$, while for $b > 1$ we have a strict inclusion $S(b, q, p) \subset DS(b, q, p)$. We also prove that $S(b)$-trees form a shrinking hierarchy by parameters b and q, that is

- if $b' < b$ then $S(b, q, p) \subset S(b', q, p)$.

- if $q' < q$ then $S(b, q, p) \subset S(b, q', p)$.

$DS(b)$-trees form the analogous hierarchy only by the parameter q.

The main result of the paper is the lower bound of utilization, which holds both for $S(b)$-trees and $DS(b)$-trees. For an n-node $S(b)$-tree T its utilization $\Delta(T) = \frac{\textit{total weight of keys in } T}{np}$.

We prove that for any $\varepsilon > 0$ the three parameters b, q, and p can be chosen in such a way that $\Delta(T) > 1 - \varepsilon$ for any tree $T \in S(b, q, p)$ with sufficiently large number of nodes n.

Finally, we construct logarithmic–time algorithms representing the three basic operations for balanced trees.

Our former papers consider a number of special cases of $S(b)$-trees [12] – [17].

$S(b)$-trees have been implemented in three software systems. The *Starset programming language* uses $S(1)$-trees described in [12] for representing its set data types [5].

A modified variant of $S(1)$-trees was used as a base model for a Linux file system originally called *TreeFS* and currently known as *ReiserFS*. The representation of a whole file system by a balanced tree has been proven to be more efficient compared to the traditional file systems both in terms of the execution time of file operations, especially for small files, and with respect to the disk space utilization.

The third system *STreeLib* [17] is an implementation of the data structure described here as a stand–alone multi–process multi–thread library written in C++. STreeLib is designed as a storage solution for semi–structured data. Particulalrly, as a data storage for native XML databases it represents an alternative to traditional B^+-trees.

2. Basic definitions

2.1. Trees

We consider trees that store keys chosen from a finite key set K. Each node $S = \langle S_0, k_1, S_1, \ldots, k_m, S_m \rangle$ of the tree contains a sequence of keys k_i from K separating references to child nodes S_i, such that if the number of keys is m then the number of references is $m + 1$. For the leaf nodes all the references are empty.

As usually we say that F is a *parent* or an *ancestor* of S if there is a path from F to S in the tree. S is called a *child* or a *descendant* of F in the case. If the path from F to S has length 1 then F is called the *direct ancestor of* S. If $F = \langle S_0, k_1, S_1, \ldots, k_m, S_m \rangle$ and $S = S_i$ then S is called the *i-th direct descendant of* F.

The number of keys m in a tree node S is called the *order of the node*. The number of the node's **non–empty** direct descendants is its *out–degree*. The out–degree of a leaf is 0, while for an internal node it equals $m + 1$. Note that here we do not accept nodes with a mixture of empty and non–empty references.

Let a path of length n lead from the tree root T to some node S. Then n is said to be the *level* of S in the tree. We suppose that larger values of n correspond to *higher* levels of the tree.

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

Let $k(S)$ denote the set of keys directly contained in S, and let $K(S)$ denote the total set of keys contained in the sub–tree rooted at S.

2.2. Structured trees

Definition 2.1 *Let $(K, \ll)$ be a finite ordered key set. A tree T is called* structured *iff*

1. *All paths in T from the root to the leaves have equal length,*
2. *For any node $S = \langle S_0, k_1, S_1, \ldots, k_m, S_m \rangle$*

$$k_1 \ll \ldots \ll k_m \;\&$$
$$\forall i (1 \leq i \leq m \Rightarrow (\forall l \in K(S_{i-1}))$$
$$(\forall r \in K(S_i))(l \ll k_i \ll r))$$

A natural number q is an *order of a structured tree* T if for each non-root vertex of the tree its order is at least q.

2.3. B-trees

Definition 2.2 *Let $q > 0$ be a natural number. A structured tree T of order q is called a* B-tree of order q *iff for any vertex S its order $|k(S)| \leq 2q$*

A *utilization* of an n-vertex B-tree T of order q is given by the ratio $\delta(T) = \frac{|K(T)|}{2qn}$

It is well known for B-trees [8], [19], [18], [9] that

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

1. $\delta(T) > \frac{1}{2} - \frac{1}{2n}$,
2. *search, insertion, and deletion of a key in T can be performed in time $O(\log n)$.*

2.4. $DS(b)$-trees

Consider a *weighted ordered set of keys* $(K, \ll, \mu)$, 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 $D \subseteq K$. The *weight of set* D is given by

$$\mu(D) = \sum_{k \in D} \mu(k)$$

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

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

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

k is called the *delimiting key* for the neighboring nodes L and R. Informally, two vertices L and R of the same level of the tree are the neighbors, if there are no other vertices of the same level between them. Note that the delimiting key for any pair of neighboring nodes belongs to the nearest common ancestor of the neighbors.

A sequence $\sigma = S_0, k_1, S_1, \ldots, k_m, S_m$ 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*. The *weight of the sweep* is defined by

$$\mu(\sigma) = \mu(S_0) + \mu(k_1) + \mu(S_1) + \cdots + \mu(k_m) + \mu(S_m)$$

A sweep σ of length m of a rank p structured tree T is said to be *dense* iff its weight

$$\mu(\sigma) > mp$$

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

Definition 2.3 *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 a $DS(b)$-*tree of order q and rank p *iff its parameters b, q, and p are natural numbers, such that*

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

The class of such trees is denoted by $DS(b, q, p)$. Note that if $q_1 \geq q_2$, then $DS(b, q_1, p) \subseteq DS(b, q_2, p)$.

2.5. $S(b)$-trees

Given a collection $S_0, S_1, \ldots, S_m$ of trees and a collection $k_1, \ldots, k_m$ of keys we can construct a new tree $S = \langle S_0, k_1, S_1, \ldots, k_m, S_m \rangle$, which is represented by the root node, consisting of the sequence of keys $k_1, \ldots, k_m$ separating $m + 1$ references to the roots of the initial trees $S_0, S_1, \ldots, S_m$.

We need another tree constructor, which given the two collections of trees and keys builds a new tree denoted $S = [S_0, k_1, S_1, \ldots, k_m, S_m]$, which is obtained by combining all the root nodes S_i separated by 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

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

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

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

Any $(m + 1)$-partition of S is determined by a sequence, of m keys from S, which uniquely specify 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:

$$\mu(S_i) \leq p \;\&\; |k(S_i)| \geq q$$

Otherwise, the partition is said to be *improper*.

A sweep $\sigma = S_0, k_1, S_1, \ldots, k_m, S_m$ of length m of a structured tree T of order q and rank p is said to be *incompressible* (with respect to p and q), iff any m-partition of $[\sigma]$ is improper. That is, if the sweep consisting of $m + 1$ nodes cannot be compressed into a smaller number of nodes retaining the original rank p and order q.

Finally, T is called b-*locally incompressible*, iff each of its sweeps of length b is incompressible.

Definition 2.4 *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 natural numbers, such that*

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

Let $S(b, q, p)$ denote the class of $S(b)$-trees (read as sweep–b–tree) of order q and rank p.

2.6. Utilization

Consider a structured tree T of order p. The ratio of the total weight $M(T)$ of the given tree T to the maximal possible weight np of an n-vertex structured tree of rank p

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

is called a *utilization* of the tree.

Note that our utilization Δ defined for structured trees composed of variable length keys differs essentially from the utilization δ known for B-trees in that Δ is a function of the weight of the tree, rather than of the number of its keys as in the case of B-trees. Δ becomes equivalent to δ only if the weight function μ identically equals 1 on the key set K.

3. A Hierarchy of Balanced Trees

For $b = 0$ both the density and the incompressibility properties are degenerate, since they don't impose any additional restrictions on a structured tree, meaning that any structured tree of rank p is both a $S(0)$-tree and a $DS(0)$-tree of rank p.

Proposition 3.1 *The class of all structured trees coincides with the union*

$$\bigcup_{q>0} \bigcup_{p>2q} S(0, q, p)$$

B-trees form a subclass of this class.

Proposition 3.2 *Let $(K, \ll, \mu_0)$ be an ordered key set with the weight function μ_0 that identically equals 1 on K. Then the class of B-trees of order q coincides with the class $S(0, q, 2q)$.*

A class of 2-3-trees ([8], [9], [18], [19], [20]) being a class of B-trees of order $q = 1$ coincides with $S(0, 1, 2)$.

Another example of $S(0)$-trees, called *B-trees with bounded key length*, is given in [12], [8]. This is the most simple variant of trees intended to store variable weight keys.

Proposition 3.3 *The class of B-trees with bounded key length of rank p is a proper subclass of the class $S(0, 1, p)$.*

For $b = 1$ the classes of $S(b)$-trees and $DS(b)$-trees also coincide. The difference between density and incompressibility manifests itself when the locality parameter $b > 1$.

Lemma 3.4 *Let S be a vertex of a structured tree such that $|S| \geq b(q+1)-1$ and $b \leq q \leq \frac{p}{2q\mu_{\max}(K)}$. Then $\mu(S) \leq bp$ implies that there exists a (p, q)-proper b-partition of vertex S.*

Using Lemma 3.4 we prove

Proposition 3.5

$$\begin{array}{lcl} S(0,q,p) & = & DS(0,q,p) \\ S(1,q,p) & = & DS(1,q,p) \\ S(b,q,p) & \subset & DS(b,q,p) \text{ for all } b > 1 \end{array}$$

Proposition 3.5 (3) means that for $b > 1$ incompressibility is a stronger property than density.

Lemma 3.6 *If a sweep σ of length b is incompressible then any of its sub-sweeps of length $b' < b$ is incompressible too.*

Lemma 3.6 directly implies that $S(b)$-trees form a shrinking hierarchy by the locality parameter b.

Proposition 3.7 *Let $b' < b \leq q \leq \frac{p}{2q\mu_{\max}(K)}$. Then*

$$S(b, q, p) \subset S(b', q, p)$$

An analogous hierarchy for $DS(b)$-trees cannot be constructed. One can easily construct two examples showing both that a sweep is dense, while its sub-sweeps are not dense, and conversely that a sweep is not dense even if its sub-sweeps are all dense.

Both $S(b)$-trees and $DS(b)$-trees form a shrinking hierarchy by their order q.

Proposition 3.8 *Let $b \leq q' < q \leq \frac{p}{2q\mu_{\max}(K)}$. Then*

$$\begin{array}{lcl} S(b,q,p) & \subset & S(b,q',p) \\ DS(b,q,p) & \subset & DS(b,q',p) \end{array}$$

As we mentioned above the properties of density and incompressibility degenerate in case of $b = 0$. Therefore, from now on we will always suppose that $b > 0$.

$DS(b)$-trees will be used for constructing lower bounds of utilization of the trees, since for proving the bounds only sweep density is used rather than incompressibility. Incompressibility is required for constructing efficient algorithms of search, insertion, and deletion in $S(b)$-trees. Finally, all the results are applied to $S(b)$-trees.

4. Space Lower Bounds

In this section we analyze space efficiency of $S(b)$-trees. All lower bounds below will be proven for $DS(b)$-trees, which by Proposition 3.5 also hold for $S(b)$-trees.

Lemma 4.1 *Let T be a structured tree of order q. Let $n > q + 1$ be the number of vertices of T. Then the number of leaves x of T is bounded by*

$$x > \frac{q(n-1)+1}{q+1}$$

Proof. This lemma will be proven by induction by the height of the tree. If the height of T is 1, then the number of leaves in it is $x = n - 1$, and we should just prove that $n - 1 \geq \frac{q(n-1)+1}{q+1}$. Indeed, by the hypothesis of the lemma $n \geq q + 2 > 2$ and thus

$$\frac{q(n-1)+1}{q+1} = (n-1) - \frac{n-2}{q+1} \leq n - 1 = x$$

Suppose now that the height of T is h, and that for all trees of height less than h the bound for the leaves has already been established. Consider the subtrees of T of height $h - 1$. Let the number of such subtrees be d each having n_i vertices and x_i leaves ($i = 1, \ldots, d$), respectively. Then

$$\begin{array}{c} n = 1 + \sum n_i \\ x = \sum x_i \end{array}$$

Using the assumption of the induction we get

$$\begin{array}{rcl} x & = & \sum_{i=1}^{d} x_i \geq \frac{1}{q+1} \sum_{i=1}^{d} (qn_i + 1) = \\ & = & \frac{1}{q+1}(q(n-1)+d) \end{array}$$

As in a structured tree $d > 1$ the required follows. □

Theorem 4.2 *Let $T \in DS(b, q, p)$ be an n-vertex tree such that $n > q + 1$. Then*

$$\Delta(T) > \frac{b}{b+1}\frac{q}{q+1} - \frac{b+1}{n}$$

Proof. Let us consider a sweep Ω of T composed of all leaves of the tree with their delimiting keys. Note that the sweep includes all keys contained in the tree. From this sweep we choose a maximal number s of disjoint sub-sweeps σ_i ($i = 1, \ldots, s$) of length b.

Since sweeps of length b in T are dense, and since all the sweeps chosen are disjoint we have

$$M(T) = \mu(\Omega) \geq \sum_{i=1}^{s} \mu(\sigma_i) > bps$$

Let x be the number of leaves of T. Then the number of chosen sub-sweeps is $s = \left\lfloor \frac{x}{b+1} \right\rfloor$. Applying Lemma 4.1 we obtain

$$\begin{array}{rcl} s & > & \frac{x}{b+1} - 1 > \frac{1}{b+1}\frac{q(n-1)+1}{q+1} - 1 > \\ & > & n\frac{1}{b+1}\frac{q}{q+1} - \frac{1}{b+1}\frac{q-1}{q+1} - 1 \end{array}$$

We know that $M(T) > bps$, and consequently

$$\begin{array}{rcl} \Delta(T) & = & \frac{M(T)}{np} > b\frac{s}{n} > \\ & > & \frac{b}{b+1}\frac{q}{q+1} - \frac{1}{n}(\frac{b}{b+1}\frac{q-1}{q+1} + b) \end{array}$$

Since $q \geq b > 0$ we obtain

$$\Delta(T) > \frac{b}{b+1}\frac{q}{q+1} - \frac{1+b}{n} \quad \square$$

This theorem implies a number of important corollaries.

Corollary 4.3 *Let the locality parameter $b > 0$ be fixed. Then for any $\varepsilon > 0$ two parameters $q \geq b$ and $p \geq 2q\mu_{\max}(K)$ can be chosen, such that for any tree $T \in DS(b, q, p)$ having $n \geq (b+1)(q+1)$ vertices its utilization*

$$\Delta(T) > \frac{b}{b+1} - \varepsilon$$

From the practical viewpoint $S(1)$-tree model is the most important modification of $S(b)$-trees. We studied the cases of $S(1)$-trees and $S(2)$-trees in details in [14] and [16] using another method of proving the bounds. The lower bound for these special cases can be obtained from Corollary 4.3 by substituting the appropriate values of the locality parameter $b = 1, 2$.

From the formal viewpoint the most interesting result gives the following

Theorem 4.4 *Let $(K, \ll, \mu)$ be a weighted ordered key set. Then for any $\varepsilon > 0$ three parameters $b > 0$, $q \geq b$ and $p \geq 2q\mu_{\max}(K)$ can be chosen, such that for any tree $T \in DS(b, q, p)$ having $n \geq (b+1)^2$ vertices its utilization*

$$\Delta(T) > 1 - \varepsilon$$

5. Algorithms

The detailed algorithms can be found in [17].

Going from the root to the leaves of the tree *Search* performs local searches in the tree nodes. It either finds the given key in the current node S or chooses a direct descendant of S and proceeds with it.

Insertion and *Deletion* both start with the search looking for a node where the given key should be either inserted or removed. In both cases it is possible to locate a leaf node for that purposes. Then a common for insertion and deletion procedure of balancing is called.

Starting from the modified leaf and moving backwards along the path the leaf was originally accessed by the search, the BALANCE procedure consecutively performs local balancing of the vicinities of the vertices in the path. For each current vertex S BALANCE decides whether S is imbalanced by verifying the following three conditions

1. one of the $b + 1$ sweeps of length b, containing S, is not incompressible.
2. $|S| < q$.
3. $\mu(S) > p$.

The conditions are verified in the order they are listed. If none of them holds, the procedure skips the level.

If (1) a compressible sweep is found in the vicinity of S then balancing of S is performed by compressing the sweep(s). Namely, BALANCE redistributes keys between the nodes of the sweep in order to empty and then to remove at least one of them. It can be proven that not more than two vertices must be removed in order to restore the balance under the condition.

If (2) all sweeps in the vicinity are incompressible but the number of keys in S is insufficient, then BALANCE merges S with one of its neighbors. If the neighbor's weight after that is normal ($\leq p$) then balancing is finished. Otherwise, the overweight neighbor becomes the current node and is further balanced under Condition 3.

Finally, (3) if everything is fine with S with respect to its order and the incompressibility of the vicinity, but the weight is not right BALANCE tries to move as many keys of S as possible to the left and/or to the right parts of the vicinity. If all the keys of S have been redistributed among the nodes of the vicinity then the node is eliminated. If it was possible to move just enough keys out of S into the neighbors, such that the weight of keys S retained is less than p then balancing is completed. Otherwise, S is split into 2 or more vertices. It can be shown that only a constant number of new vertices can appear that way for each level.

Balancing of the current level of the tree can modify some parent vertices, namely the direct ancestor of S and its two nearest neighbors, but never affects the higher-level

nodes that have been balanced before. The modified parent vertices are balanced in their turn when the procedure reaches them moving backwards along the path.

Finally, BALANCE stops at the tree root. If the root turned out to be empty, it releases the root. If the root exceeds the required weight, it creates a new root and splits the old one into two or more nodes as above.

Thus the procedure examines all the nodes that lay on the path from the root of T to the leaf it started with and balances them if necessary together with the vicinities of the path nodes. Each vicinity consists of at most $2b+1$ vertices, which means that the total number of nodes modified during balancing is bounded by $h(2b+1)$, where h is the height of T.

The time complexity of the algorithms is summarized by

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

References

[1] G.M. Adel'son-Vel'skii, E.M. Landis, *An Algorithm for the Organization of Information*, Soviet Math. Doklady vol. 3, 1972, pp. 1259–1262.

[2] R. Bayer, *Symmetric binary B-tree: Data Structure and Maintenance Algorithms*, Acta Inf., vol. 1, 4, 1972, pp. 290–306.

[3] R. Bayer, E. McCreight, *Organization and Maintenance of Large Ordered Indexes*, Acta Inf., vol. 1, 3, 1972, pp. 173–189.

[4] C.C. Foster, *A Generalization of AVL-Trees*, Communications of the ACM, vol. 16, 1973, pp. 513–517

[5] M.M. Gilula, *The Set Model for Database and Information Systems*, Addison-Wesley (In Association with ACM Press): Wokingham, 1994.

[6] L.J. Guibas, R. Sedgewick, *A Dichromatic Framework for Balanced Trees*, Proceedings, 19-th Annual IEEE Symposium on Foundations of Computer Science, 1978, pp. 8–12

[7] G.K. Gupta, B. Srinivasan, *Approximate Storage Utilization of B-tree*, Inf. Proc. Lett. vol. 22, 1986, pp. 243–246

[8] D.E. Knuth, *The Art of Computer Programming*, vol. 3 (Sorting and Searching), Addison–Wesley, Reading, MA, 1973.

[9] H.R. Lewis, L. Denenberg, *Data Structures and Their Algorithms*, HarperCollins, NY, 1991.

[10] E.M. McCreight, *Pagination of B^*-Trees with Variable-Length Records*, Commun. ACM, vol. 20, 9, 1977, pp. 670–674.

[11] A.L. Rosenberg, L. Snyder, *Time– and Space–Optimality in B-Trees*, ACM Trans. Database Syst. 6, 1, 1981, pp. 174-193.

[12] A.P. Pinchuk, K.V. Shvachko, *Maintaining Dictionaries: Space-Saving Modifications of B-Trees*, Lecture Notes in Computer Science, vol. 646, 1992, pp. 421–435.

[13] K.V. Shvachko, *Space-Saving Modifications of B-Trees*, In Proceedings of Symposium on Computer Systems and Applied Mathematics, St.Petersburg, 1993, p. 214.

[14] K.V. Shvachko, *$S(1)$-trees: Space Saving Generalization of B-Trees with 1/2 Utilization*, 1994 (unpublished).

[15] K.V. Shvachko, *Optimal Representation of Dynamic Dictionaries by Balanced Trees*, In Proceedings of XI International Conference on Logic, Methodology, and Philosophy of Science, Obninsk, Russia, vol. 2, 1995, pp. 181-186 (in Russian).

[16] K.V. Shvachko. *Space Saving Generalization of B-Trees with 2/3 Utilization*, Computers and Mathematics with Applications, vol. 30, No.7, 1995, pp. 47-66.

[17] K.V. Shvachko, *$S(b)$-tree Library: An Efficient Way of Indexing Data*, DIMACS: Series in Discrete Mathematics and Theoretical Computer Science, vol. 50, External Memory Algorithms (J.M. Abello, J.S. Vitter, Eds.), American Mathematical Society, DIMACS, 1999.

[18] T.J. Teorey, D.P. Fry, *Design of Database Structures*, vol. 2, Prentice-Hall, Englewood Cliffs, NJ, 1982.

[19] N. Wirth, *Algorithms and Data Structure*, Prentice-Hall, Englewood Cliffs, NJ, 1986.

[20] D. Wood, *Data Structure, Algorithms, and Performance*, Addison-Wesley Publishing Company, 1993.

[21] A.C.-C. Yao, *On Random 2-3 Trees*, Acta Inf., vol. 9, 1978, pp. 159–170.