B-Skip Trees, a Data Structure between Skip Lists and B-Trees

Joaquim Gabarró
Xavier Messeguer

Report LSI–94–48–R

B-Skip trees, a data structure between Skip lists and B trees *

Joaquim Gabarró Xavier Messeguer †

December 14, 1994

Abstract

At a first look a skip-list is rather a collection of smartly connected linear linked list than a tree but they are, however, closely connected to trees. To prove it, we introduce random B-Skip trees that inherits the performance rates of skip-lists. Moreover we give a bijection between both data structures that commute with elementary operations.

Random B-Skip trees are randomized B trees where the number of keys of an internal node is given by a geometrically distributed random variable with parameter p. A random B-Skip tree with n keys and parameter p has a $O(\log_{1/p} n)$ expected height and $(1-p)/p$ expected number of keys in a node, consequently an update operation can be done in expected time $O(\log_{1/p} n)$ time. The expected number of split and join operations needed to insert or delete a key is independent of n and is equal to $1/(1-p)$.

Keywords Skip-lists , B trees, dictionaries, randomized data structures.

1 Introduction

As we will consider a data structure close to Skip-lists and B trees, let us start giving a short description of them. B trees are balanced trees introduced by Bayer and McCreight in 1972 [3], see also [6, 5]. A B tree has lower and upper bounds on the number of keys in a node. These bounds are expressed in terms of a fixed integer $t \geq 2$ called *minimum degree*. The number of keys in an internal node is between $t+1$ and $2t+1$. Keys and child pointers are located alternately and all the leaves have the same depth. To simplify the proofs we assume all the keys to be different. Let us recall the search, insert and delete operations on B trees with the conventions adopted along this paper. We keep all the operations local:

- The *search* of a key in a B tree is a top down process. In one step the key is compared with the keys located into the current node and sent down to the corresponding child.
- To *insert* a key, first we search the place of insertion. Second, a new leaf with the key is *hung* to the corresponding internal node of height 1 (leaves have height 0 and the root has maximal height) and the key is located between the keys of this node. Third, if the number of keys exceeds $2t+1$, the node is *split* and the key is sent up as many times as necessary.

*This research was supported by the ESPRIT BRA Program of the EC under contract no. 7141, project ALCOM II.

†Departament de Llenguatges i Sistemes Informàtics.Universitat Politècnica de Catalunya.Pau Gargallo 5, 08028-Barcelona, Spain.Contact e-mail: gabarro@lsi.upc.es

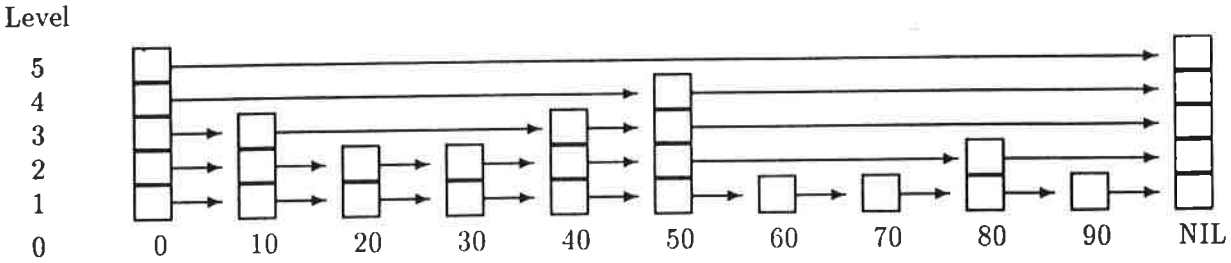


Figure 1: Skip List

- To *delete* a key, first we search the unique internal node (all the keys are different) having this key. Second, we move this key down *joining* two children and putting the key between them and we iterate until the key is located into the node having height 1. Four, take off the key of this node and *unhang* the leaf.

If the tree has n leaves, the number of *splits* and *joints* in the preceding algorithms is bounded by the height of the tree $O(\log_t n)$.

We considers also Skip-lists introduced by W. Pugh in 1990 [13]. They are randomized data structures composed by a collection of connected linear linked lists every one having a different level (see the figure 1). All the items of a Skip List are stored in the list of level 1. Some of them belongs also to the list of level 2 and so forth. To decide which item should be included in the list of next level a random device is used. The acces to the data is determined by two pointers. Let be x a key contained in the list of level $l > 0$, then it has a pointer to the next greather key in the same level, denoted *forward pointer*, and a pointer to the equal key of the level $l - 1$, denoted *down pointer*.
Let us recall the search, insert and delete algorithms on Skip-lists . We adopt a description inspired in algorithms on concurrent Skip-lists given by W.Pugh [12]. As before we keep al the operations local:

- To *search* a key into an skip list we do a top down process. Given a node level (x, l) we compare the key with the key in forward(x, l) and we move the key forward or down. The process stops at level zero.

- To *insert* a key, first we search the place of insertion. Second, we determine the level l of the new node using a geometrically distributed random variable with parameter p. Third, we *hang* the key at level 1. Four, we move *upward* the level of the node, inserting it in all the lists with level smaller level or equal to l.

- To *delete* a key, first we search the node containing that key. Second, we move *downward* the level of it, deleting succesively the key in all the lists with level smaller or equal to l. At level 1 we *unhang* the key.

In W. Pugh [13] it is proved that, a skip list with parameter p and n keys, has $O(\log_{1/p} n)$ expected number of linked list and the expected time to update a key is $O(\log n)$. Later on, S. Sen [14] show that performance deviates from the expected time decreases with probability $O(n^{-2})$. Moreover the expected height of a key is $1/q$.

The B-Skip trees inherit from B trees the locality of the operations. They are B trees where the number of keys in a node is determined by a geometrically distributed random variable with parameter p. It is in this aspect that B trees are also close to Skip-lists . In fact we can prove a bijection between both data structures. Recall that mappings between data structures are a well-known topic, see for instance R. Bayer [2] show that every AVL is essencially a 2–3–4 tree. Later on L. J. Guibas, R. Sedgewick [8] embed in a dichromatic frame the most used balanced tree schemes and T. Papadakis in his PhD thesis [10] introduce deterministic Skip-lists and give us a bijection between these lists and 2–3 trees. Finally, B-Skip trees have the same performance rates than random search trees by Aragon and Seidel [1]. However random search trees should be applied into different context, because the probability of a key is given by a continous identically distributed random variable. Therefore two keys does not have the same probability.

The paper is divided into five sections. The second one give a definition of B-Skip trees and its basic oparations. The third one relates B-Skip trees and Skip-lists giving a bijection between both data estructures. The next one analyses algorithms on random B-Skip trees. The last section contains some possible extensions.

2 B-skip trees

The B-Skip trees (figure 2) share many of the aspects of B having two main differences. First, we allow internal nodes without keys (white nodes in the figure) having only one child. These nodes just propagate information without any addressing. Second, the number of keys in a node does not have an upper or lower bound. As B-Skip trees will be random structures, later on, we will prove that the number of keys in internal nodes depends on the distribution of probability and it is independent of the number of keys. We follow the notations given in [5]

Definition 1 *A* B-Skip T *is a rooted tree having the following properties:*

1. *Every internal node* x *has three registers,* height(x) *gives us the height of the node,* key(x) *stores a possible empty ordered list of keys and* child(x) *stores an ordered list of pointers.*
2. *If the internal node does not have any key,* child(x) *has a unique child.*
3. *If* key(x) *has keys* $a_1 < a_2 < \ldots < a_p$, child(x) *contains* $p+1$ *pointers to its children. Moreover, the keys separate the ranges of the keys stored in each subtree: If* b_i *is any key stored in the* i*-subtree we have* $b_0 < a_1 \leq b_1 < a_2 \leq b_2 \ldots b_{p-1} < a_p \leq b_p$.
4. *Every leaf* l *constains one key and the information associated with this key in the registers* key(l) *and* inf(l).
5. *The leaves does not have any children and all of them have the same depth.*

The split operation is well known in the context of B trees. Here, we extend it to B-Skip trees. To do it we need to deal adequately with white nodes (figure 3). To simplify we assume all the keys in T are different. Given a B-Skip tree T and a key a belonging to node x of T, we define:

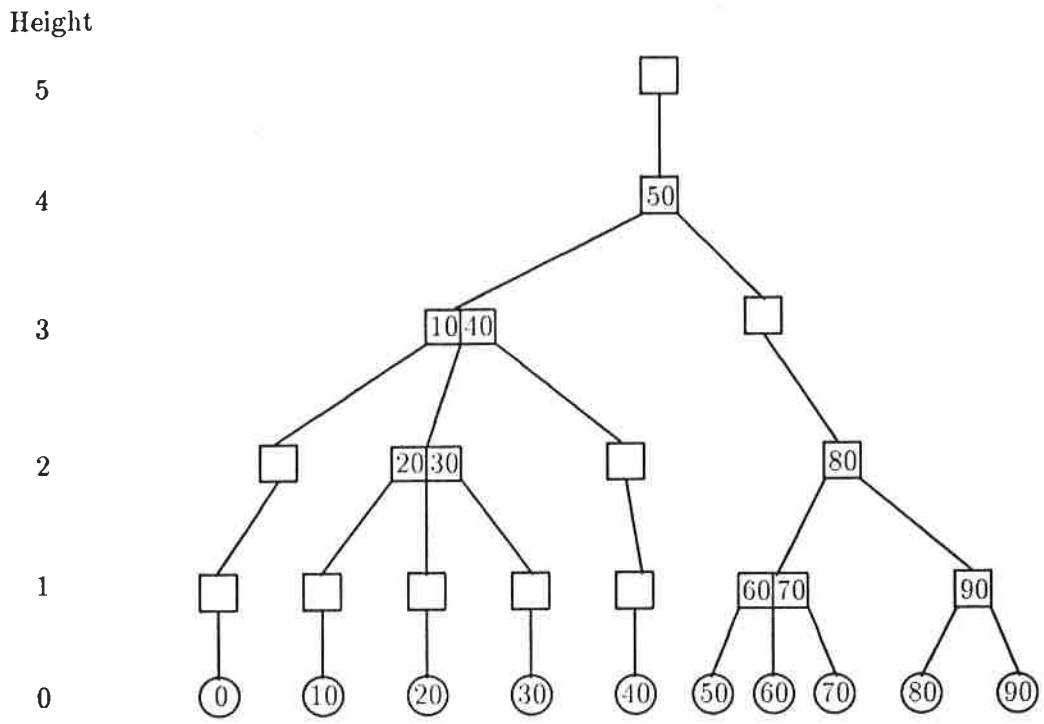


Figure 2: B-Skip tree

$SPLIT(T, a)$ = " if $x \neq root(T)$, the node x is split into two other ones and
the key a is located into the father of x,
if $x = root(T)$ a new node will be created to memorize x,
in both cases, when necessary white nodes will be created or propagated".

Sometimes we will note $SPLIT(T, a)$ as $SPLIT_a(T)$. The different context of a into T give us different types of split. Enumerate them. When the father of x is not a white node we have three cases:

- The key a has left and right brothers. The split does not generate any white node.
- The key a has only left or right brothers. The split generates a white node.
- When the key a has no brothers at all, the split generates two white nodes.

When the father of a is a white node, we have also three cases:

- When a has left and right brothers. The split take off the white node.
- If a has only left or right brothers, the split send down the white node.
- When the key a has no brothers at all, the white node is splited and propagated down.

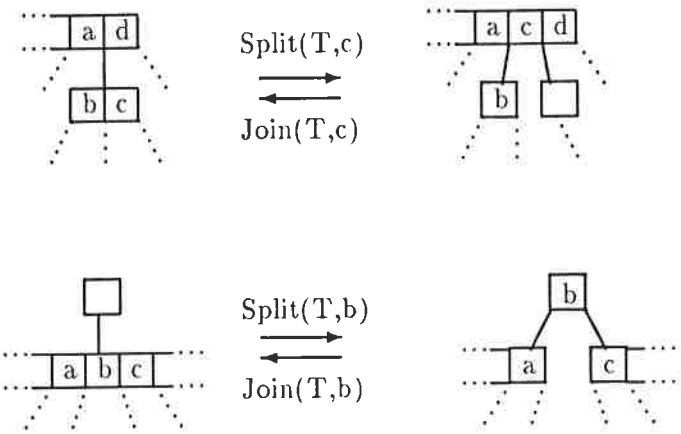


Figure 3: Splits and Joins with white nodes

The figure 3 schematizes two of the preceding cases.

Dually, we can define the inverse operation $JOIN(T, a)$. This operation, groups together the internal nodes located at both sides of the key a into only one internal node and putting a between them. In $JOIN(T, a)$ the height of a decreases by one. It follows:

$$JOIN_a(SPLIT_a(T)) = SPLIT_a(JOIN_a(T)) = T$$

Remark that, both operations, $SPLIT$ and $JOIN$, deal with keys belonging to T. We can iterate both operations, we note $SPLIT_a^l(T, a)$ the consecutive application of l split operations onto the key a. Dually we can define $JOIN_a^l(T, a)$. Moreover, we need an operation hanging a new key a into T.

$HANG(T, a)$ = "it hangs a new leaf with value a into T at height 0
and locates the key a adequatelly at level 1. That means,
if the node is white, fulfill it with a otherwise insert
a between the keys located on this internal node".

As before we can, easily define the unhang of a leaf if this key has height 1. We note this operation of $UNHANG(T, a)$. If we like to hang a leaf with value a and locate the key a at height $l > 1$ we do

$$SPLIT_a^{l-1}(HANG_a((T)).$$

Reciprocally, if we like to delete a leaf a having a key at level $l > 1$ we can do

$$UNHANG_a(JOIN_a^{l-1}(T)).$$

3 B-skip trees and Skip lists

There is a close connection between B-Skip trees and Skip-lists because Skip-lists can be easily transformed into B-Skip trees. To deal with this transformation let us recall some

basic notations in Skip-lists [7]. Given a skip list S and a node $x \neq \mathbf{NIL}$ and some integer $0 \leq l \leq \text{level}(x)$, we write

$$\begin{aligned} \text{wall}(x,l) \;=\; & \text{"the first node } y \text{ to the right of } x \text{, i.e.} \\ & \text{key}(x) < \text{key}(y), \text{ such that level}(y) > l\text{"}. \end{aligned}$$

For instance, in Figure 1, wall(header(S), 3) is the node having key 50. Moreover the *subskiplist at* (x,l) of S, denoted $S(x,l)$ or $S_{x,l}$ for short, is the skip list of height l, where x acts as a header and wall(x,l) acts as **NIL**. Node/levels in $S_{x,l}$ are those reachable from (x,l).

Before to give an accurate definition of:

$$\mathcal{T} : \text{Skip-lists} \longrightarrow \text{B-Skip}$$

let us explain this top-down transformation informally, see figure 4

There are three main cases:

- The first case happens when we have a subskiplist $S_{x,l}$ such that $0 < l < \text{level}(x)$ and forward(x,l) = wall(x,l) (dashed lines on the picture indicates $l < \text{level}(x)$ and $l < \text{level}(\text{wall}(x,l))$). The transformed B-Skip tree $\mathcal{T}(S_{x,l})$ start with a white node of height l having as child $\mathcal{T}(S_{x,l-1})$. Remark that the smallest key of $\mathcal{T}(S_{x,l-1})$ will be be key(x).

- The second case corresponds to $l > 0$ and $l < \text{level}(x)$ and forward$(x,l) \neq$ wall(x,l). Therefore there is at least one node between forward(x,l) and wall(x,l). We schematize two subcases. When there is only one node y_1 between forward(x,l) and wall(x,l), the B-Skip tree $\mathcal{T}(S_{x,l})$ has an internal node with key $a_1 = \text{key}(y_1)$ and two children corresponding to the transformations of $S_{x,l-1}$ and $S_{y_1,l-1}$. If there are two nodes y_1 and y_2 between forward(x,l) and wall(x,l), the B-Skip will has an internal node with keys $a_1 = \text{key}(y_1)$, $a_2 = \text{key}(y_2)$ and three children corresponding to $S_{x,l-1}$, $S_{y_1,l-1}$, $S_{y_2,l-1}$. In general, if forward$^f(x,l)$ = wall(x,l), there will be $y_1, y_2, \ldots y_{f-1}$ nodes between (x,l) and the wall, and $\mathcal{T}(S_{x,l})$ starts having a node with f children corresponding to $S_{x,l-1}, S_{y_1,l-1}, \ldots S_{y_{f-1},l-1}$ and $f-1$ keys $a_1 = \text{key}(y_1), \ldots, a_{f-1} = \text{key}(y_{f-1})$.

- Case 3. The last case happens when we have $l = 0$ or $S_{x,0}$. Therefore we get a leaf having the key a of x.

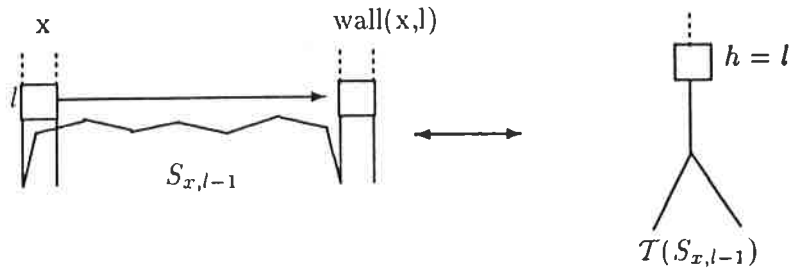
In a formal way we define the $\mathcal{T}$ as:

Definition 2 *Given a skip list* $S = S(\text{header}, \mathbf{NIL})$ *we define* $\mathcal{T}(S)$ *recursively starting from* $\mathcal{T}(S(\text{header}, \mathbf{NIL}))$. *Given a node/level* (x,l) *we consider the integer* $f \geq 1$ *such that* forward$^f(x,l)$ = wall(x,l), *therefore*

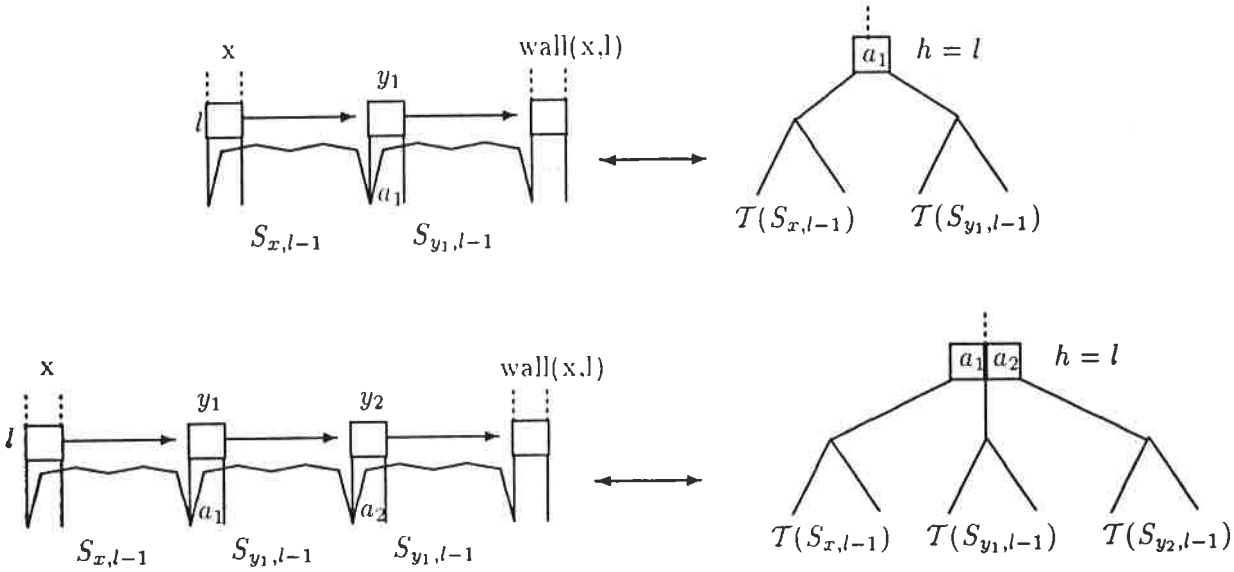
1. *When* $f = 1$ *and* $l > 0$ *we have*

$$\mathcal{T}(S_{x,l}) \;=\; \begin{array}{c} \square \\ \downarrow \\ \mathcal{T}(S_{x,l-1}) \end{array}$$

Case 1 : $0 < l < \text{level}(x)$ and $\text{forward}(x, l) = \text{wall}(x, l)$



Case 2 : $0 < l < \text{level}(x)$ and $\text{forward}(x, l) \neq \text{wall}(x, l)$



Case 3 : $l = 0$

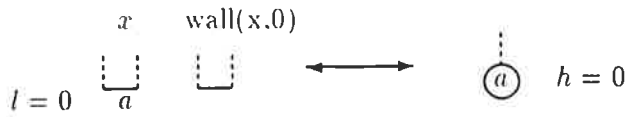


Figure 4: Mapping $\mathcal{T}$ between Skip-lists and B-Skip trees

2. *When* $f > 1$ *and* $l > 0$ *we note* y_i =forward$^i(x,l)$ *(in particular* $x = y_0$ =forward$^0(x,l)$*),* $a_i = \text{key}(y_i)$ *and* $T_i = \mathcal{T}(S_{y_i,l-1})$ *we have:*

$$\mathcal{T}(S_{x,l}) = \begin{array}{ccccccccc} [& a_1 & | & a_2 & | & \cdots & | & a_{f-1} &] \\ \swarrow & & \downarrow & & \downarrow & & \downarrow & & \searrow \\ T_0 & & T_1 & & T_2 & & T_{f-2} & & T_{f-1} \end{array}$$

3. *When* $l = 0$ *we have*

$$\mathcal{T}(S(x,l)) = \begin{array}{c} \downarrow \\ \text{key}(x) \end{array}$$

We can easly verify that Skip List given in the figure 1 transforms into the B-Skip tree of the figure 2.

Lemma 1 *The mapping* $\mathcal{T}$: Skip-lists $\longrightarrow$ B-Skip *is a bijection.*

We call $UPWARD(S,a)$ the operation on Skip-lists corresponting to the $SPLIT(T,a)$ operation of B-Skip trees. This operation appears implicitely in the work of W. Pugh [12] on concurrent Skip-lists . As before, assume that all the keys in S are different, therefore we can associate a unique node x to a key a such that $a = \text{key}(x)$. Informally, this operation corresponds to increase the level of node x by one.

$UPWARD(S,a)$ = "increase the level of x by one and recontruct S, when level(x) = level(S), increase level(S) by one".

As before we note also $UPWARD(S,a) = UPWARD_a(S)$. When level$(x) > 0$ we can define appropiately the inverse operation $DOWNWARD(S,a)$, decreasing the level of x by one, in such a way that:

$$DOWNWARD_a(UPWARD_a(S)) = UPWARD_a(DOWNWARD_a(S)) = S$$

As we have made before with the B-Skip trees, we introduce the operations $HANG_a(S)$ and $UNHANG_a(S)$ hanging a key in a node of level 1 and unhanging a key belonging to a node of level 1. More interesting is that operations on B-Skip tress matches with operations on Skip-lists . More precisely:

Lemma 2 *Given a skip list* S *having all the keys different, there is the following relationship between operations on skip list* S *and the* B-Skip *tree* $\mathcal{T}(S)$*:*

1. *Given a keys* a *belonging to* S *it holds* $\mathcal{T}(UPWARD_a(S)) = SPLIT_a(\mathcal{T}(S))$.
2. *Given a key* a *wich does not appears in* S*, it holds* $\mathcal{T}(HANG_a(S)) = HANG_a(\mathcal{T}(S))$.

Proof. First, give us a proof outline of (1). Assume that $a = \text{key}(x)$ and $l = \text{level}(x)$. As in $UPWARD_a(S)$ the level of x will be $l+1$, we consider in S the smallest subskip S' containing x with level$(S') > l$. When x goes up one level it will cut the forward pointer at level $l+1$ of a node y belonging to S'. We would need to consider three main cases depending on y and z =forward$(y, l+1)$.

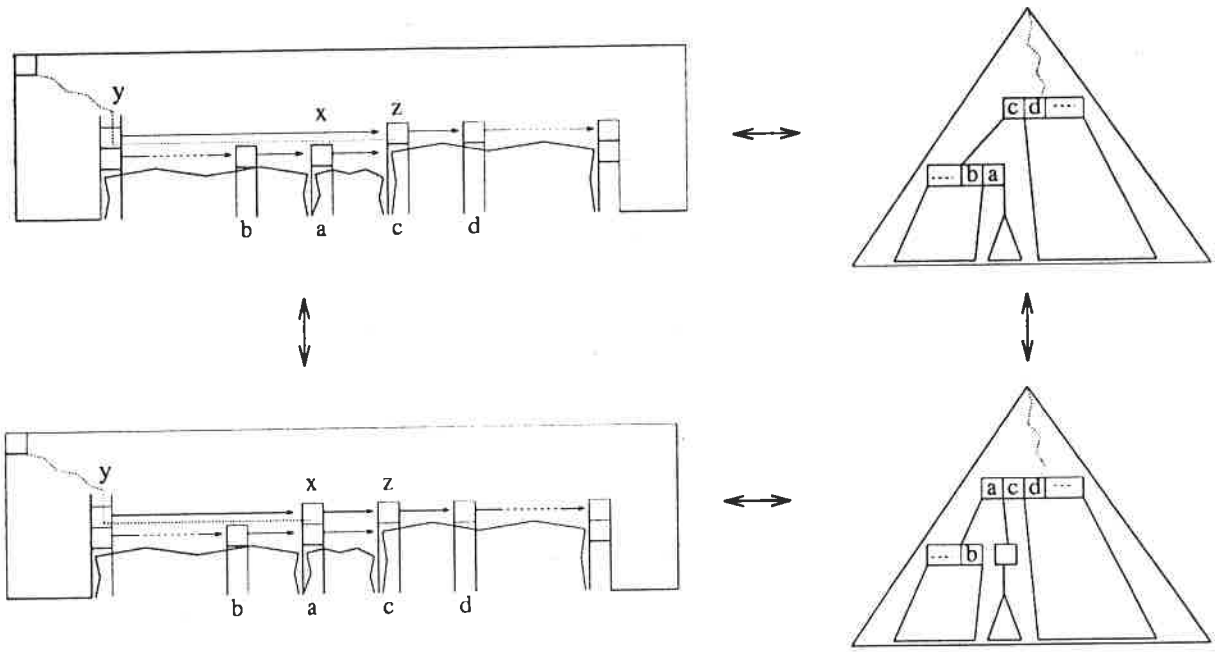


Figure 5: Commutation of $SPLIT$ and $UPWARD$

- Both y and z verify level(y) = level$(z) = l + 1$.
- Both y and z verify level$(y) > l + 1$ and level$(z) > l + 1$.
- The node y verifies level$(y) > l + 1$ but node z verifies level$(z) = l + 1$.
- Reciprocally we have, level$(y) = l + 1$ but node z verifies level$(z) > l + 1$.

For every one of these cases there are another four possibilities, depending on the brothers surrounding x.

- The node x is surrounded by left and right brothers having exactly level l.
- The node x has only a left brother with level exactly l.
- The node x has only a right brother having level exactly l.
- There are no brothers surrounding x and having level l.

The preceding considerations allow us a total of 16 different context for x inside S'. The figure 5 schematizes the equality $\mathcal{T}(UPWARD_a(S)) = SPLIT_a(\mathcal{T}(S))$ in the case where x has only a left brother of level l. All the other 11 cases and the case (2) follow the same lines. ■

We can deal in a similar way with the inverse operations. Therefore, given a skip list S with different keys, and a key a with level greather than one, it holds:

$$\mathcal{T}(DOWNWARD_a(S)) = JOIN_a(\mathcal{T}(S)).$$

If the case a has level one, we have $\mathcal{T}(UNHANG_a(S)) = UNHANG_a(\mathcal{T}(S))$.

4 Algorithms on random B-Skip trees

We develop search, insert and delete procedures for B-Skip trees. These procedures are quite similar to those given in the introduction for Skip-lists and B trees. The height of the tree will be determined by a random variable $H = 1 + NB(1, q)$ being $NB(1, q)$ the negative binomial random variable with parameter q, also called Pascal or geometrical distribution (as the level of Skip-lists). This distribution is equal to the number of failures seen before a succes in a serie of independents trials, where the probability of succes in a trial is q. The expected value of $NB(1, q)$ is p/q being $p = 1 - q$, therefore the expected value of H is $1/q$.

- *Procedure search:* It is similar to the search procedure of B trees, the only difference is given by the existence of white nodes. When the key a to be search points to a white node b moves down, otherwise a points to an internal node with keys $b_1, \ldots, b_l$ and the key a moves right until it finds the adecuate place to move down. This action is executed while the height of the current node is positive.

- *Procedure insert:* It is randomized and it has four steps. First, using a search, we found in T the place to insert a. Second, the key is pended to T executing $HANG_a(T)$. Third, the value $h = height(a)$ is generated following a $1 + NB(1, q)$ random variable. Four, the new key is sent up executing $SPLIT_a^{h-1}(T)$. From the preceding lemmas we get $\mathcal{T}(INSERT_a(S)) = INSERT_a(\mathcal{T}(S))$.

- *Procedure delete:* To delete a key a, first we search the internal node having the key a. Second, if the internal node has height $h > 1$, we send down this key executing $JOIN_a^{h-1}(T)$ procedure. Finally the node is erased with $UNHANG_a(T)$. We get $\mathcal{T}(DELETE_a(S)) = DELETE_a(\mathcal{T}(S))$.

In the preceding section we give a rather combinatorial definition of B-Skip trees. However, we are interested in the set of trees generated from the empty tree inserting and deleting a collection of keys by the above procedures, these trees will be randomized data structure (like skip lists). Formally, we define:

Definition 3 *We call a* B-Skip *tree T "random with parameter p" if it has been generated through a set of insertions (with parameter p) and deletions.*

These trees inherits from Skip-lists the following properties:

Theorem 1 *Let be T a random* B-Skip *tree with parameter p and size n ($q = 1 - p$):*

1. *The expected height of T is $O(\log_{1/p} n)$. Moreover, the probability that the expected height deviates significantly from the expected value decreases as $O(n^{-2})$.*

2. *The expected number of keys in an internal node is q/p. Moreover, the probability that this expected number deviates c times from the expected value q/p decreases as $q^{cq/p}$.*

Proof. (1) Given a random tree T with n keys, the skip list $S = \mathcal{T}^{-1}(T)$ is a usual skip list with n keys where the height of the nodes is determined by a random variable $1 + NB(1, q)$

and S has height $O(\log_{1/p} n)$. To prove that the expected height decreases exponentially we apply the Chernoff tail bound lemma [4],

$$\text{Prob}\{H \geq a\} \leq E(e^{tH})/e^{ta} \qquad \forall a, t > 0.$$

The expectations for of H are $E(e^{tH}) = q/(e^{-t} - p)$ being $q = 1 - p$. Then

$$\text{Prob}\{H \geq c \log_{1/p} n\} \leq \frac{q}{e^{-t} - p} / e^{tc \log_{1/p} n} = \frac{q}{p^{2/c} - p} n^{-2}$$

taking $t = \frac{2}{c}\ln(1/p)$.

(2) Let G the random variable whose value is the number of keys in a node of a tree T. G is equal to the number of consecutive nodes with the same level in $S = \mathcal{T}^{-1}(T)$, therefore $G = NB(1, p)$ and its expected value is q/p (this magnitude is denoted *gap* by Papadakis [10]) and:

$$\text{Prob}\{G > cE(G)\} = \text{Prob}\{G > cq/p\} = \sum_{k \geq cq/p} \text{Prob}\{G = k\} = \sum_{k \geq cq/p} pq^k = q^{c\frac{q}{p}}$$

■

The following theorem show us that the B-Skip trees have very interesting performances to deal with dictionaries. Therefore are an interesting alternative to Skip Lists and B trees. As before, given a random tree T, the inverse $S = \mathcal{T}^{-1}(T)$ is a skip list. Therefore we can easly translate the results obtained in Skip-lists and B and:

Theorem 2 *Let be T a* B-Skip *tree of size n and random parameter p.*

1. *The expected time to search, insert or delete a key is $O(\log_{1/p} n)$ and the probability that the expected height deviates significantly from the expected value decreases as $O(n^{-2})$.*
2. *The expected number of splits in an insertion or joins in a deletion $1/q$.*

5 Practical considerations and extensions

The B trees handle the index of a large data base. Instead B trees we propose to use B-Skip trees. This substitution can be interesting when the height of the tree and the number of keys in a node can be suitable bounded. A random variable X is "suitable bounded" when $\text{Prob}\{X > cE(X)\}$ is "small".

Lemma 3 *Let be a random* B-Skip *tree of size n and parameter p, let G and H be random variables giving the number of keys in an internal node and the height of the tree:*

1. $\text{Prob}\{G > cE(G)\} \simeq (eq)^{-c}$ *for small values of p.*
2. $\text{Prob}\{H > 1 + L(n)\} = 1 - (1 - \frac{p}{n})^n \simeq 1 - e^{-p}$ *for greather values of n.*

Proof. (1) Recall that $\text{Prob}\{G > cE(G)\} = \left(q^{q/p}\right)^c$, expression that can be approximated by

$$q^{\frac{q}{p}} = (1-p)^{\frac{1-p}{p}} = \frac{(1-p)^{1/p}}{(1-p)} < (qe)^{-1}$$

(2) Recall that the height of each node is also a negative binomial but with parameter q: $H = NB(1,q)+1$. Then:

$$\text{Prob}\{H_n > L(n)+1\} = 1 - \text{Prob}\{H_n \leq 1+L(n)\} = 1 - \prod_{k=1}^{n} \text{Prob}\{H \leq 1+L(n)\}$$

$$= 1 - \prod_{k=1}^{n} \left(1 - p^{L(n)+1}\right) = 1 - (1-p^{L(n)+1})^n = 1 - \left(1 - \frac{p}{n}\right)^n \simeq 1 - e^{-p}$$

because they are a independent random variables. ■

Small values of p and the bigger values of n appears when we would like to replace B trees by B-Skip trees. If the degree of the B tree is t and the number of keys in a node is approximately $1.5t$ we take a B-Skip tree with $p \simeq 1/(1.5t)$. The probability that one node grows more than $2t$ is $\text{Prob}\{G > cE(G)\} \simeq 0.26$ for $2t \geq 90$ and $c \simeq \frac{4}{3}$. But that is the probability that only *one* node become larger than $2d$. As the number of keys in a node does not depends from the keys of the remainder nodes, the probability that k different nodes have a big number of keys decreases exponentially as $0.26^k \geq 2^{-k}$, and the number of this kind of nodes becomes negligible. Also, the probability that the height of the B-Skip tree become $\log_{1/p}(n) + 1$ is approximately 1% with the same value of p and for values of n greather that 10000 items. Therefore, we think that for this range of values, B-Skip trees are a good alternative to B trees.

From another point of view, it seems to be easy extend sequential algorithms on B-Skip trees to massively parallel versions. The main lines seems to mimics the works done in 2–3 trees by W. Paul, U. Viskin and H. Wagener [11], in B trees by L. Higham and E. Schenk [9] and in Skip-lists by J. Gabarró, C. Martínez and X. Messeguer [7]. Also, the massively parallel approach inherits the advantages of skiplists and trees.

Acknowledgements

We thank Conrado Martínez to help us with this research.

References

1. C. Aragon and R. Seidel. Randomized search trees. In *Proc. of 30th Symposium on Foundations of Computer Science*, pages 540–545, 1989.

2. R. Bayer. Symmetric binary B-trees: data structure and mantenance algorithms. *Acta Informatica*, (1):290–306, 1972.

3. R. Bayer and C. McCreight. Organization and maintenance of large ordered indexes. *Acta Inf.*, 1(3):173–189, 1972.

4. H. Chernoff. A measure of asymptotic efficiency for tests of hypothesis based on the sum of observations. *Ann. Math. Statist.*, 23:493–507, 1952.

5. T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms.* McGraw Hill, MIT, 1990.

6. C. Douglas. The ubiquitous B-tree. *Computing Surveys*, 11(2):121–137, 1979.

7. J. Gabarró, C. Martínez, and X. Messeguer. Parallel update and search in skip lists. In R. Baeza-Yates, editor, *Computer Science 2: Research and Applications*, pages 15–26. Plenum Press, New York, 1994. To appear, a complete version, in TCS.

8. L. Guibas and R. Sedgewick. A dichromatic framework for balanced trees. In IEEE, editor, *Proc. of 19th Symposium on Foundations of Computer Science*, pages 8–21, 1978.

9. L. Higham and E. Schenk. Maintaining B-trees on an EREW PRAM. *J. Parallel and Distributed Computing*. 22:329–335. 1994.

10. T. Papadakis. *Skip Lists and Probabilistic Analysis of Algorithms.* PhD thesis, University of Waterloo, 1993. Available as Technical Report CS-93-28.

11. W. Paul, U. Vishkin, and H. Wagener. Parallel computation on 2-3 trees. In J. Díaz, editor, *Proc. 10th International Colloquium on Automata, Programming and Languages, LNCS 154*, pages 597–609. Springer-Verlag, 1983.

12. W. Pugh. Concurrent maintenance of skip lists. Technical Report CS-TR-2222.1, Institute for Advanced Computer Studies, Department of Computer Science, University of Maryland, College Park, MD. Apr 1989. Also published as UMIACS-TR-90-80.

13. W. Pugh. Skip lists: a probabilistic alternative to balanced trees. *Comm. ACM*, 33(6):668–676, 1990.

14. S. Sen. Some observations on skip-lists. *Inform. Proc. Lett.*, 39(3):173–176, 1991.

Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya

Research Reports – 1994

LSI–94–1–R "Logspace and logtime leaf languages", Birgit Jenner, Pierre McKenzie, and Denis Thérien.

LSI–94–2–R "Degrees and reducibilities of easy tally sets", Montserrat Hermo.

LSI–94–3–R "Isothetic polyhedra and monotone boolean formulae", Robert Juan-Arinyo.

LSI–94–4–R "Una modelización de la incompletitud en los programas" (written in Spanish), Javier Pérez Campo.

LSI–94–5–R "A multiple shooting vectorial algorithm for progressive radiosity", Blanca Garcia and Xavier Pueyo.

LSI–94–6–R "Construction of the Face Octree model", Núria Pla-Garcia.

LSI–94–7–R "On the expected depth of boolean circuits", Josep Díaz, María J. Serna, Paul Spirakis, and Jacobo Torán.

LSI–94–8–R "A transformation scheme for double recursion", José L. Balcázar.

LSI–94–9–R "On architectures for federated DB systems", Fèlix Saltor, Benet Campderrich, and Manuel García-Solaco.

LSI–94–10–R "Relative knowledge and belief: SKL preferred model frames", Matías Alvarado.

LSI–94–11–R "A top-down design of a parallel dictionary using skip lists", Joaquim Gabarró, Conrado Martínez, and Xavier Messeguer.

LSI–94–12–R "Analysis of an optimized search algorithm for skip lists", Peter Kirschenhofer, Conrado Martínez, and Helmut Prodinger.

LSI–94–13–R "Bases de dades bitemporals" (written in Catalan), Carme Martín and Jaume Sistac.

LSI–94–14–R "A volume visualization algorithm using a coherent extended weight matrix", Daniela Tost, Anna Puig, and Isabel Navazo.

LSI–94–15–R "Deriving transaction specifications from deductive conceptual models of information systems", María Ribera Sancho and Antoni Olivé.

LSI–94–16–R "Some remarks on the approximability of graph layout problems", Josep Díaz, María J. Serna, and Paul Spirakis.

LSI-94-17-R "SAREL: An assistance system for writing software specifications in natural language", Núria Castell and Àngels Hernández.

LSI-94-18-R "Medición del factor modificabilidad en el proyecto LESD" (written in Spanish), Núria Castell and Olga Slávkova

LSI-94-19-R "Algorismes paral·lels SIMD d'extracció de models de fronteres a partir d'arbres octals no exactes" (written in Catalan), Jaume Solé and Robert Juan-Arinyo.

LSI-94-20-R "Una paral·lelització SIMD de la conversió d'objectes codificats segons el model de fronteres al mdel d'octrees clàssics" (written in Catalan), Jaume Solé and Robert Juan-Arinyo.

LSI-94-21-R "Clausal proof nets and discontinuity", Glyn Morrill.

LSI-94-22-R "A formal method for the synthesis of update transactions in deductive databases without existential rules", Joan A. Pastor.

LSI-94-23-R "On the sparse set conjecture for sets with low density", Harry Buhrman and Montserrat Hermo.

LSI-94-24-R "On infinite sequences (almost) as easy as π", José L. Balcázar, Ricard Gavaldà, and Montserrat Hermo.

LSI-94-25-R "Updating knowledge bases while maintaining their consistency", Ernest Teniente and Antoni Olivé.

LSI-94-26-R "Structural facilitation and structural inhibition", Glyn Morrill.

LSI-94-27-R "Formalising existential rule treatment in the automatic synthesis of update transactions in deductive databases", Joan A. Pastor.

LSI-94-28-R "Proceedings of the Fifth International Workshop on the Deductive Approach to Information Systems and Databases" (Aiguablava, 1994), Antoni Olivé (editor).

LSI-94-29-R "Towards a VRQS representation", Mariona Taulé Delor.

LSI-94-30-R "MACO: Morphological Analyzer Corpus-Oriented", Sofía Acebo, Alicia Ageno, Salvador Climent, Javier Farreres, Lluís Padró, Francesc Ribas, Horacio Rodríguez, and Oscar Soler.

LSI-94-31-R "Lexical mismatches: a semantic representation of adjectives in the LKB", Salvador Climent and Carme Soler.

LSI-94-32-R "LDB/LKB integration", German Rigau, Horacio Rodríguez, and Jordi Turmo.

LSI-94-33-R "Learning more appropriate selectional restrictions", Francesc Ribas.

LSI-94-34-R "Oracles and queries that are sufficient for exact learning", Nader H. Bshouty, Richard Cleve, Ricard Gavaldà, Sampath Kannan, and Christino Tamon.

LSI–94–35–R "Acquisition of lexical translation relations from MRDs", Ann Copestake, Ted Briscoe, Piek Vossen, Alicia Ageno, Irene Castellón, Francesc Ribas, German Rigau, Horacio Rodríguez, and Anna Samiotou.

LSI–94–36–R "The query complexity of learning context-free grammars", Carlos Domingo and Víctor Lavín.

LSI–94–37–R "Learning minor closed graph classes with membership and equivalence queries", John Shawe-Taylor, Carlos Domingo, Hans Bodlaender, and James Abello.

LSI–94–38–R "On the integration of CAD and CAE", C.M. Hoffmann and R. Juan-Arinyo.

LSI–94–39–R "Logic of assertions", Ton Sales.

LSI–94–40–R "Between logic and probability", Ton Sales.

LSI–94–41–R "A sequential and parallel implementation of skip lists", Xavier Messeguer.

LSI–94–42–R "Higher-order linear logic programming of categorial deduction", Glyn Morrill.

LSI–94–43–R "A language for constructive parametric solid modelling", Lluis Solano and Pere Brunet.

LSI–94–44–R "A note on genericity and bi-immunity", José L. Balcázar and Elvira Mayordomo.

LSI–94–45–R "On RNC approximate counting", Josep Díaz, María J. Serna, and Paul Spirakis.

LSI–94–46–R "Prototipatge semàntic d'un model conceptual deductiu" (written in Catalan), C. Farré and M.R. Sancho.

LSI–94–47–R "Generació i simplificació automàtica del Model d'Esdenivents Interns corresponent a un model conceptual deductiu" (written in Catalan), C. Farré and M.R. Sancho.

LSI–94–48–R "B-Skip trees, a data structure between skip lists and B trees", Joaquim Gabarró and Xavier Messeguer.

Copies of reports can be ordered from:

Nuria Sánchez
Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
Pau Gargallo, 5
08028 Barcelona, Spain
secrelsi@lsi.upc.es