

Dynamic Optimality for Skip Lists and B-Trees

Prosenjit Bose *

Karim Douieb^{†§}

Stefan Langerman^{‡§}

Abstract

Sleator and Tarjan [39] conjectured that splay trees are dynamically optimal binary search trees (BST). In this context, we study the skip list data structure introduced by Pugh [35]. We prove that for a class of skip lists that satisfy a weak balancing property, the working-set bound is a lower bound on the time to access any sequence. Furthermore, we develop a deterministic self-adjusting skip list whose running time matches the working-set bound, thereby achieving dynamic optimality in this class. Finally, we highlight the implications our bounds for skip lists have on multi-way branching search trees such as B-trees, (a - b)-trees, and other variants as well as their binary tree representations. In particular, we show a self-adjusting B-tree that is dynamically optimal both in internal and external memory.

1 Introduction

The comparison-based *dictionary problem* is one of the most fundamental in information processing: How do we store and retrieve data efficiently? Consider a totally ordered set S of n elements each associated with a key. A dictionary is a data structure that allows the following operations:

- Search(x): Find the element x in the set S .
- Insertion(x): Add the element x to S .
- Deletion(x): Remove the element x from S .

It is well known that in the comparison-based model, the cost of these operations has a lower bound of $\lceil \log(n+1) \rceil$ in the worst case. Since the AVL-tree [1], numerous alternative solutions to the dictionary problem have been developed that guarantee worst case or expected $\Theta(\lg n)^1$ cost per operation.

The dictionary problem becomes different if one is interested in the performance of a structure over a sequence of operations as opposed to just the worst case time of a single operation. In this paper, we are interested in understanding the optimal time needed for a data structure to perform an *access sequence* $X = \{x_1, x_2, \dots, x_m\}$ of search operations in the comparison model of computation on a pointer machine. The operations consist of several basic *unit-cost operations* such as rotations, promotions/demotions or splits/joins depending on the structure considered.

We define a data structure by specifying a model of computation that describes the structure of the information itself and the unit-cost operations allowed. We define the minimum number of unit-cost operations needed for a data structure in model M to access sequence X as $OPT_M(X)$. Thus, the optimal *offline* access algorithm, i.e., the algorithm which knows every access of the sequence in advance, executes X in $OPT_M(X)$ unit-cost operations. An *on-line* algorithm, which serves each access without any knowledge of future accesses, is said to be *dynamically optimal* if it executes X in time $O(OPT_M(X))$. More generally, we say that an algorithm A is $f(n)$ -*competitive* if its access cost $A_M(X)$ does not exceed $f(n)OPT_M(X)$.

Without any prior knowledge of the access sequence, data structures must be *self-adjusting* to be competitive, i.e., they not only adapt during insertions or deletions of elements but also during searches. In the next section we review some of the relevant literature related to our results.

Before presenting our results, we define the *Working-set Property*: In an access sequence, the *working-set number* $t_i(z)$ of an element z at time i is the number of *distinct* elements accessed since the last access of z prior to time i , including z . A data structure has the *working-set property* if the amortized cost of access x_i is $O(\lg t_i(x_i))$. Over an access sequence X , we define the working set bound as $WS(X) = \sum_{i=1}^m \lg t_i(x_i)$.

In this paper, we analyze the relationship between several models of skip lists and multi-way search tree data structures. We give lower bounds on the optimal time needed to perform any access sequence X in these models. We show that for some of these models, the

^{*}School of Computer Science, Carleton University, Ottawa, Ontario, K1S 5B6. jit@scs.carleton.ca
Research supported in part by NSERC.

[†]Boursier FRiA, kdouieb@ulb.ac.be

[‡]Chercheur qualifié du FNRS, stefan.langerman@ulb.ac.be

[§]Département d'Informatique, Université Libre de Bruxelles, CP212, Boulevard du Triomphe, 1050 Bruxelles, Belgium.

¹ $\lg x$ in this paper is defined as $\log_2(x+2)$.

working-set bound is equivalent to dynamic optimality. We also develop a deterministic self-adjusting skip list in this specific model that guarantees an upper bound on the access cost matching the working set, i.e., it is dynamically optimal. We extend our results to show dynamic optimality of B-trees [8] both in internal and external memory under split and join operations.

In Section 3, we formally define several models of self-adjusting data structures. Lower bounds on optimal access cost for some skip list models are provided in Section 4. In Section 5, a dynamically optimal deterministic self-adjusting skip list is developed. In Section 6, we highlight the implications our bounds for skip lists have on B-trees. In particular, we show a self-adjusting B-tree that is dynamically optimal for B constant. Finally in Section 7, we show how our results on skip lists lead to dynamically optimal B-trees in external-memory.

2 Prior Work

The linked list (LL model in Section 3) is one of the first self-adjusting data structure to be analyzed. Deterministic on-line algorithms were studied for this model such as *Move-To-Front* (MTF) which simply moves the accessed element to the front of the list. It was shown by Sleator and Tarjan [38] that the MTF strategy is 2-competitive. An unpublished result of Karp and Raghavan [27] states that no deterministic algorithm is c -competitive, where $c < 2$. The Bit algorithm [36], which moves to front the accessed element with probability $1/2$, is expected to be 1.75-competitive. The best randomized algorithm so far, due to Albers [3], is 1.6-competitive. No randomized list update algorithm can be better than 1.5-competitive [43]. It is NP-hard to compute an optimum offline strategy [4].

The binary search tree (BST model in Section 3) is the first efficient data structure to have been formally studied more specifically in the case where we are just allowed to perform rotations. Two lower bounds on $OPT_{BST}(X)$ were given by Wilber [45]. Both are complex functions of X . A slight variation of the first one, the *interleaves* lower bound, is given by Demaine *et al.* [15]. Recently Derryberry *et al.* [16] presented a graphical interpretation of $OPT_{BST}(X)$, the *rectangle cover* lower bound, which generalizes both the interleaves lower bound and Wilber's second lower bound. Determining whether those lower bounds are tight is still open.

So far the *splay tree* [39], introduced by Sleator and Tarjan, is the only known BST algorithm which may be $O(1)$ -competitive. This constitutes the *dynamic optimality conjecture*. The splay tree has numerous properties described by the Balanced Theorem [39], the

Static Optimality Theorem [39], the Static Finger Theorem [39], the Working-Set Theorem [39], the Sequential Access Theorem [18, 40, 41, 42] and the Dynamic Finger Theorem [12, 13]. In [23], splay trees have been shown to be $O(1)$ -competitive against a class of dynamic balanced binary search trees. Alternatives were introduced by Demaine *et al.* [15], followed by Derryberry *et al.* [17] and Georgakopoulos [24] to achieve $O(\lg \lg n)$ -competitive BST algorithms.

The multi-way branching search tree (MWBST model in Section 3) is an abstract data structure model that generalizes B-trees [8], $(a-b)$ -trees, 2-3-trees or many of their variants. This model allows a node to contain an arbitrary number of elements and thus have an arbitrary number of children. The update operations allowed are the same as in B-trees, i.e., split and join of nodes. To the best of our knowledge, lower bounds on the optimal access sequence cost in this tree model have not been studied before.

Others Structures inspired but differing from the MWBST model have been adapted in a self-adjusting fashion. The splay tree of Sleator and Tarjan [39] has been generalized by Sherk [37] to k -ary trees. Martel [29] presented another self-adjusting multi-way search tree, the *k-forest*, which is a collection of different B-trees of branching factor k and of doubly exponential increasing size. Assuming a node is processed in $O(\lg k)$ unit-cost operations, a k -forest performs an access sequence X in $O(H(p))$ expected time, where $H(p)$ is the entropy of the probability distribution $p = \langle p_{s_1}, p_{s_2}, \dots, p_{s_n} \rangle$ with p_{s_j} the probability of accessing element s_j in X . Martel shows how a k -forest can be adapted to achieve worst case performance by using additional structures.

With a similar idea of using a forest of trees, Iacono [26] and later Bădoiu *et al.* [10] developed an alternative to the splay tree, not included in the BST model, with $O(\lg n)$ worst-case access times that satisfies all of the original splay tree properties proved in [39] and the Unified property.

The skip list of Pugh [35] is a probabilistic alternative to balanced trees. It is a dictionary data structure storing a dynamic set S of ordered elements and which has an expected $O(\lg n)$ running time per operation. A skip list is built in levels, the bottom level is an ordinary sorted linked list and each higher level acts as an "express lane" for the lists below (see Fig. 1). The *height* $h(s)$ of an element s is defined as the highest level where s appears. The height $H(\mathcal{L})$ of a skip list \mathcal{L} is defined as $\max_{s \in \mathcal{L}} h(s)$ and the *depth* $d(s)$ of s is $H(\mathcal{L}) - h(s)$. The pointers connecting two items on a same level and the pointers leading to the lower level of an item are called *forward* and *down* pointers

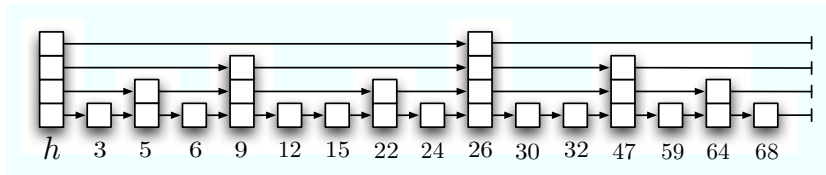


Figure 1: A skip list, with h its header.

respectively.

Several variants of the skip list have been considered: Munro *et al.* [34] developed a deterministic version of the skip list, based on B-trees [8], that performs insert, delete and search operations in worst case $O(\lg n)$ time through the increment and decrement of the height of various elements (promotion/demotion operations).

Under the assumption that the distribution of access probabilities is given, Martínez and Roura [30] developed an algorithm that minimizes the expected access time by either building an optimal static skip list in $O(n^2 \lg n)$ time or a nearly optimal one in $O(n)$ time. Later, Bagchi *et al.* [6] developed the biased skip list as suggested by Mehlhorn and Näher [31]. It manages a biased dictionary, i.e., an ordered set S of elements v associated with weight $w(v)$ and performs search, insert, delete, join, split, finger search and reweight operations in worst case running times similar to those of biased search trees [9, 22].

Skip lists cited above are not self-adjusting as they do not update themselves during a search operation. Over a sequence of accesses these structures have the following problem: Frequently accessed elements might have smaller height in the list than less frequently accessed elements, implying that it takes longer to reach frequently accessed elements. A self-adjusting skip list must somehow raise frequently accessed elements to higher levels in the list to speed up future accesses. The self-adjusting skip lists due to Ergun *et al.* [19, 20, 21] perform an operation x_i in $O(\lg t_i(x_i))$ expected time. With structural modification of the original skip list structure (not a real skip list anymore as defined by Pugh), they show in [20] that it can support insertion/deletion of x_i in $O(\lg t_i^{max}(x_i))$ expected amortized time, where $t_i^{max}(x_i)$ is the maximum working number x_i attains during its life-time prior to time i . The second self-adjusting structure due to Ciriani *et al.* [11] performs an access sequence X in $O(H(p))$ expected amortized time, where $H(p)$ is the entropy of the probability distribution $p = \langle p_{s_1}, p_{s_2}, \dots, p_{s_n} \rangle$ with p_{s_j} the probability of accessing element s_j in X . This self-adjusting skip list has been adapted for managing arbitrary long strings residing in external memory, where the cost is

measured in terms of number of disk access (I/Os) fetching pages of a fixed size B .

Note that another self-adjusting skip list presented as an exercise in the book of Mulmuley [33], maintains an access frequency count for each element and considers them as weights such that the structure performs standard operations in expected running times similar to those of biased search trees [9, 22].

Relation between skip list and tree models:

Skip lists were introduced as a probabilistic alternative to balanced trees. Several studies have shown that in fact both models are tightly related. Several authors [33, 32, 5, 34, 31, 6, 14] have noticed that a skip list can be interpreted for the same cost as a specific kind of tree and vice versa. In [34, 31, 6] a skip list simulates MWBSTs such as B-trees [8], (a, b) -trees [25] or other variant in order to obtain deterministic skip lists. Messeguer [32] took the opposite approach by developing the *skip tree*, belonging to the MWBST model, consisting of a one-to-one mapping from standard skip list operations to new randomized tree operations. Thus skip list and MWBST models have been shown to be equivalent. As suggested by Munro *et al.* [34], skip lists can be interpreted as BST by using a standard symmetric binary B-tree transformation scheme [7] which includes red-black trees [28]. Recently Dean and Jones [14] provided a detailed description of how to represent a skip list using BST. They also showed that a modification of the skip list structure (see model $SL\Delta$ in section 3) can simulate any BST based on rotation-balancing mechanisms.

3 Models

LL: In the *linked list* model defined by Sleator and Tarjan [38], each element contains pointers to the next and to the previous element respectively. A finger is initialized to the front of the list at each access, the search operation consists of moving the finger forward and backward by one position at unit-cost. Two other operations are allowed: the unit-cost *swap* of two adjacent elements and the *free swap* which moves the accessed element anywhere closer to the front.

BST: We refer here to the formal *binary search tree* model described by Wilber [45]. In this model, the accessed element must be moved to the root of the tree by doing a sequence of unit-cost rotations. Other unit-cost rotations can be done as well in order to speed up future accesses. The access cost is the total number of rotations performed plus 1.

MWBST: This model of *multi-way branching search tree* defines ordered trees where nodes are composed of several elements. Nodes have a number of children (*branching factor*) which is the number of element inside the node plus 1. To perform a standard search [8] in the MWBST model we use a finger initialized at the root of the tree after each access. Every movement of the finger from an element to its child node or to its next sibling in the node costs 1. Other unit-cost operations are allowed such as *split* and *join* of nodes touched by the finger during a search. The split operation consists of moving an element x of a node v into its parent node u , elements storing a key smaller than x remain in the original node v and a new node storing the other elements of x is created as another child of u . The join operation performs the inverse of a split operation.

SL: In the standard skip list model we use a single finger. To access x_i , the finger is initialized to the highest level of the header element. The search algorithm must reach x_i by performing several *unit-cost operations*, namely move the finger to the forward element on the current level or to the upper/lower level of the current element. Additionally the height of any element *touched* by the finger can be incremented or decremented at unit cost (possibly many times).

SL Δ : This model was defined in [14]. Although it is not used in the remainder of this paper, we present it here to highlight the differences with our models. This model is similar to the previous one except that every element x of the list stores an offset value $\Delta(x)$. During a search, when the finger is positioned on the highest level of an element x , the model allows the finger to move down by $\Delta(x)$ levels at unit-cost. The offset value of an element can be updated for free when its height is incremented or decremented.

As shown in [14], this model can be implemented by replacing the levels column of every element of the SL Δ with an array.

MTT: Here we consider an alternative model inspired by Wilber's [45] lower bounds on binary search trees. In this skip list model, the element x_i being accessed must be *moved to the top*. Namely, the height of x_i and other elements in the list must change so that x_i is directly pointed to by the header of the list. Note that during an access, any other element can be promoted or demoted which may affect the

cost of future accesses (this contrasts with the SL model where only the touched elements are involved in those operations). The access cost is precisely the total number of promotion or demotion operations performed.

We can further specify that for each skip list model, it must be either *bounded* in the sense that we never go forward more than B times without going down in a search or *weakly bounded* in the sense that the first i highest levels contain no more than $\sum_{j=0}^i B^j$ elements. When we impose the bounded or weakly bounded constraints to a model M, we refer to it as M-B or M-WB respectively. Note that trivially $OPT_M(X) \leq OPT_{M-WB}(X) \leq OPT_{M-B}(X)$.

The bounded and the weakly bounded constraints can also be generalized to the MWBST model, by bounding the branching factor from above with B or enforcing that the sum of elements in every node at depth lower than i is at most $\sum_{j=0}^i B^j$. Note that MWBST-B corresponds exactly to the B-tree [8] data structure.

4 Lower bounds for skip list models

LEMMA 4.1. *For any access sequence X , $\frac{1}{2}OPT_{MTT-WB}(X) + m \leq OPT_{SL-WB}(X)$ and $\frac{1}{2}OPT_{MTT-B}(X) + m \leq OPT_{SL-B}(X)$.*

Proof. We prove the first statement of this lemma by showing that we can simulate a sequence of k unit cost operations in a skip list in the model SL-WB by a sequence of at most $2(k-1)$ unit cost operations in a skip list in the model MTT-WB. The unit cost operations of an access in a skip list in the SL-WB model can be decomposed in two parts, the number of finger movements and the number of height changes. The number of finger movements is at least the depth $d(x_i)$ of the accessed element plus 1 if x_i is found directly by moving the finger from a level of the header or plus 2 otherwise. Now in the MTT-WB model we must move x_i to the top with a cost of either $d(x_i)$ if the position of x_i is before the element that is directly reachable from the highest level of the header or with a cost of $d(x_i)+1$ otherwise. In order to be consistent with the skip list we want to simulate in the SL-WB model, we must move the element back to its original position before the next access, which doubles the cost. Every other height modification is done in both models for the same cost. Thus the simulation of an access in the MTT-WB model requires at most twice the cost of this access minus 1 in the SL-WB model.

The proof of the second statement is identical. \square

LEMMA 4.2. For any access sequence X , $OPT_{SL-B}(X) \leq (B+2)OPT_{MTT-B}(X)$.

Proof. We want to simulate a skip list in the model MTT-B by a skip list in the model SL-B. By the definition of a skip list in MTT-B model, the length of a search path from the header to element x_i is at most B times $d(x_i)$. The skip list in SL-B can simulate the height change of every element on the search path for the same cost as in the model MTT-B. But in MTT-B, height changes does not necessarily involve elements on the search path. Thus to simulate those height changes in SL-B, we perform the following procedure: Consider the search path from the header to x_i in the skip list in SL-B. We say that an element *hits* this path if an update operation in MTT-B, performed during previous accesses, has increased its height so that the element would be touched by a forward pointer of the path. The procedure consists of walking on this search path, each time we identify an element y which hits it, we move the finger to touch y and we change its height as in MTT model. The cost of this operation in SL-B is at most $(B+2)$ times the cost of changing the height of y in MTT-B. Then we continue the walk. Elements which have not been treated yet will wait future accesses to be modified. The amortized simulation cost of a skip list in the MTT-B model by one in the SL-B is at most $(B+2)$ times its own cost, i.e., $OPT_{SL-B}(X) \leq (B+2)OPT_{MTT-B}(X)$. \square

LEMMA 4.3. For any access sequence X , $\frac{WS(X)}{\lg B} \leq OPT_{MTT-WB}(X) + m$.

Proof. Define $h'_i(z)$ the lowest height an element z reached since its last access or since the initial time and prior to time i . Let $a = h_i(x_i) - h'_i(x_i)$ be the difference between the lowest height of an access x_i and its height at time i . The amortized cost of an access x_i corresponds at least to the number of unit cost operations needed to set it at height $h'_i(x_i)$ then to raise it at $h_i(x_i)$. As we consider amortized cost, we can assume that the promotion operation on element x_i is fully paid by itself whereas the cost of the demotion operation is equally shared among x_i and each of the elements on its level. The amortized cost of an access is necessarily smaller or equal if we assume that an element x_i reaches its lowest height $h'_i(x_i)$ then only move up by a levels exactly at time i . At this precise time x_i rises above β elements of height greater or equal to $h'_i(x_i)$, by the definition of the weakly bounded constraint, we have $\beta \leq \frac{B^{a+1}-1}{B-1}$. Indeed, in a weakly bounded skip list of height $h'_i(x_i) + a$ at most $\sum_{j=0}^a B^j$ elements can be present in the first a levels.

This means that among the $\alpha \geq t_i(x_i)$ distinct elements which went above x_i since its last access, at least $\alpha - \beta$ of them have decreased their height under $h'_i(x_i)$. Thus they have charged x_i by a cost of at least $\frac{1}{\alpha} + \frac{1}{\alpha-1} + \dots + \frac{1}{\beta+1} = H_\alpha - H_\beta$, where H_k is the k th harmonic number. Indeed, if we consider that δ elements among α have already been demoted to a height which is smaller than the height of x_i then the lowest cost that x_i must pay for an element decrementing its height corresponds to $1/(\alpha - \delta)$ and occurs in a situation where $\alpha - \delta$ elements are on the same level than x_i .

Thus the total amortized access cost of x_i is at least $a + H_\alpha - H_\beta \geq \log_B \beta - 1 + H_\alpha - H_\beta \geq \ln \beta \left(\frac{1}{\ln B} - 1 \right) + \ln \alpha - 1 \geq \frac{\ln \alpha}{\ln B} - 1 \geq \log_B t_i(x_i) - 1$. \square

Surprisingly, a skip list algorithm in the SL model without the bounded constraint can perform some access sequences in time significantly faster than in the SL-B model. Take the sequential access sequence $X = \{1, 2, \dots, n\}$, according to lower bounds proved above, this access sequence takes $\Omega(n \lg n)$ time in the SL-B or SL-WB models with B constant. Whereas the following offline algorithm performs the sequence in linear time in the SL model: Beginning with every element at height 1, we increment the height of the accessed element and decrement the height of the previously accessed one, yielding each access constant. Determining if this skip list structure can perform as well in an online situations is an interesting problem which remains to be analyzed.

5 Deterministic self adjusting skip list

We now describe a skip list data structure that conforms to the SL-B model and whose running time matches the lower bound from the previous section. Skip lists can be adapted from the k -forest structure of Martel [29] or from the working set structure of Iacono [26] to guarantee same performances. A similar but randomized approach has been taken by Ferragina *et al.* [44] for constructing a self-adjusting data structure for string dictionary operations in the external memory model. We present a variation of this structure which guarantees better performance and does not require any extra information:

The skip list stores a dynamic set S of n keys. The structure is a single linked skip list of height $2^{k+2} - 2$ and composed of $2k = 2 \lceil \lg \lg n \rceil$ layers $l_1, l'_1, l_2, l'_2, \dots, l_k, l'_k$. Layers l_a and l'_a covers each 2^a levels of the list. The layers are structured in increasing order, l_a first then l'_a , such that l_1 covers the highest level of the skip list (see Figure 2). An element whose height

is within the range of levels for a layer is associated to it.

Search:

1. The first phase of a search operation i consists in performing a traditional skip list search [35] of the element x_i in the self-adjusting skip list just like in a standard skip list.
2. Assume x_i is associated with layer l_b . If $b = 1$ the search operation is finished. Otherwise we must move x_i to l_1 and perform an *overflow* procedure if necessary.

Note that moving an element from a layer to another consists of changing its height. The ordered set of elements associated with a layer is divided in contiguous subsets by the elements belonging to higher layers. Our deterministic self-adjusting skip list manages each subset of each layer as a sublist performing insertion, deletion, split and join in $O(Bh)$, where h is the maximum height of the skip lists involved in one of those operations. The $(a-b)$ -skip list of Bagchi *et al.* [6] achieves this performance worst case. It is in fact a skip list representation of $(a-b)$ -trees [25] inheriting their properties, namely, the number of consecutive elements at height i between any consecutive elements of height strictly greater than i is at least a and at most b with $1 < a \leq \lfloor b/2 \rfloor$. By setting $b = B$ the $(a-b)$ -skip list of height h guarantees operations in $O(Bh)$ worst case time and contains $O(B^h)$ elements.

We use this $(a-b)$ -skip list as a black box in our structure. Consider now the movement of an element from a layer b to a with $b > a$, this consists of inserting the element in the corresponding sublist in layer a and splitting every sublist traversed by the element during its movement from layers b to $a - 1$. The inverse movement from a to b consists of deleting the element from the corresponding sublist in layer a and joining the sublists previously divided by the element in every layer from $a - 1$ to b .

An overflow in layer l_a occurs when the number of levels covered by l_a is no longer sufficient to maintain the elements associated with it, i.e., when a sublist reaches a height larger than 2^a . An overflow procedure performed in a layer l_a consists first in moving every element associated with l'_a to l_{a+1} and second in decreasing every height of elements associated to l_a by 2^a in order to be associated to l'_a . Then the procedure is iterated in l_{a+1} if an overflow occurs in it, and so on.

THEOREM 5.1. *The self-adjusting skip list data structure described above conforms to the SL-B model and performs any access sequence X in $O(B \frac{WS(X)}{\lg B})$.*

Proof. A simple search from the highest level of the header of the list to an element x , assuming $x \in l_a$, takes $O(B2^a)$ time in worst case. Indeed, the search can be decomposed in $2a$ subsearches in each the $2a$ first layers. A subsearch in layer l_b or l'_b , with $b = 1, 2, \dots, a$, only involves elements associated with the same layer. Thus it corresponds to a traditional search in a skip list of height 2^b . The total search cost is $O(B \sum_{b=1}^a 2^b) = O(B2^a)$. This also holds when searching for an element in layer l'_a . Thus the whole search procedure of element x associated with layer l_a or l'_a has a cost of $O(B2^a)$.

Now we show that the search cost of an element z with a working-set number $B^{2^{a-1}} < t_i(z) \leq B^{2^a}$ is $O(B2^a)$: The number of operations which are needed to overflow an empty layer l_b is at least B^{2^b} . Thus this greatest layer in which an element z , originally in l_1 , may be associated with after B^{2^a} accesses is l_a . Thus the search cost of z in the SL-B model is given by $O(B2^a)$. The cost of moving z to level l_1 is also $O(B2^a)$.

A cascade of overflows can occur after this move, and the cost of an access must integrate that. We will see that the cost of those operations corresponds, in an amortized sense, to the search cost of an element. We define the non negative potential of an element z to be B times its height in the skip list, i.e., $\Phi(z) = Bh(z)$. Thus an element involved in an overflow procedure has enough potential to pay for moving to a lower layer. It is easy to see that by increasing their cost by only a constant factor, the access operations pay for the increase in potential of the elements. \square

THEOREM 5.2. *For any access sequence X , $\frac{WS(X)}{2 \lg B} \leq \frac{1}{2}(OPT_{MTT-WB}(X) + m) \leq OPT_{SL-WB}(X) \leq OPT_{SL-B}(X) \leq cB \frac{WS(X)}{\lg B}$ with c constant.*

Proof. Trivially deduced from Lemma 4.1, 4.2, 4.3 and Theorem 5.1. \square

COROLLARY 5.1. *Our self-adjusting skip list is dynamically optimal in the SL-B model with B constant.*

The structure can be maintained dynamically. Insertion and deletion operations are straightforwardly deduced from the access approach, and they have a $O(\lg n)$ cost.

6 B-tree

As discussed in the introduction, skip lists in the SL-B model can be represented in the B-trees (or MWBST-B) model using the detailed reduction of Dean and Jones [14]. Thus identical upper and lower bounds follow immediately such that we readily obtain the following:

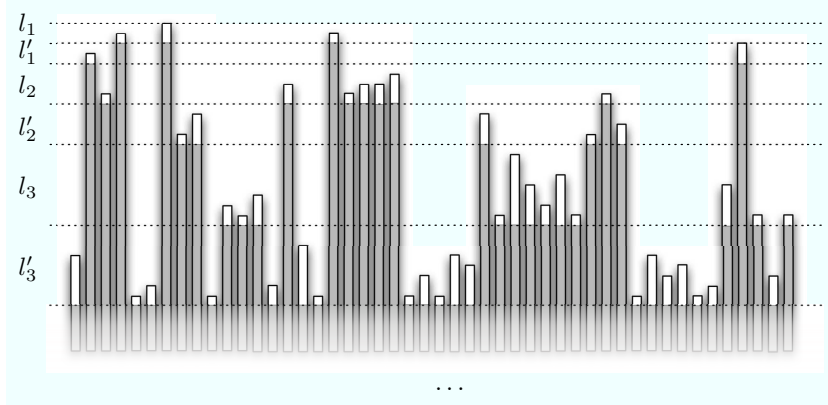


Figure 2: Deterministic self-adjusting skip list.

COROLLARY 6.1. *There exists a self-adjusting B-tree that is dynamically optimal with B constant.*

7 B-tree in external-memory model

The *external-memory model* [2] (also known as the I/O model) considers two-levels of memory hierarchy, the cache and the disk each divided in blocks of elements of size B . The cache has a limit of M blocks and the disk has an unbounded size. We can only access the memory block loaded in the cache for free cost, if the desired information is not in the cache then the memory block containing it in the disk is transferred into the cache. When the cache becomes full, some blocks are transferred back to the disk to make space for new block. Those transfers from disk to cache or inversely are defined as unit-cost operations.

The B-trees are well adapted to the external-memory model, assuming the size B of the memory block known, the maximum number of elements inside a node is fixed to B . In this case, an entire node can be transferred in one transfer operation. If we search for an element in a B-tree, we achieve an information-theoretically optimal number of memory transfers of $O(\log_B n)$ in worst case. When studying the running time of access sequences, it is assumed that the cache is empty at the beginning of every access. The external version of the B-tree model, I/O-B-tree, thus allows moving the finger to a child, split and join operations at unit cost. But unlike in the B-tree (or MWBST-B) model, moving the finger to a sibling is free.

We recall that in the internal-memory model (studied in previous section), the unit cost operations of MTT-B are promotion and demotion of elements, the horizontal movements of the finger is for free and the consecutive number of these movements is bounded

by B . Thus the correspondence between MTT-B in internal-memory and B-trees in external-memory is easily established. Indeed, the split and join operations in a B-tree correspond to the promotion/demotion operations in MTT-B and consulting elements inside a node is free. We can show $OPT_{MTT-B}(X) + m \leq 2OPT_{I/O-B-tree}(X)$ for any access sequence X using an identical argument as in Lemma 4.2. Then by Lemma 4.3, we have $\frac{WS(X)}{\lg B} \leq 2OPT_{I/O-B-tree}(X)$.

In the self-adjusting skip list developed in Section 5, we have subsets of elements associated with a layer which are each surrounded by two elements belonging to higher layers. Those subsets are managed with the structure of Bagchi *et al.* [6], that is a skip list simulating an $(a-b)$ -tree. Thus if we fix the value b to B (with $a \leq b/2$), we obtain a self-adjusting skip list where we have $\Theta(B)$ consecutive elements at height i between two elements of greater height. Consider that, in external memory, those consecutive elements are in the same memory block. Then any operation in a subset of elements of size $O(B^h)$ and height $O(h)$ is performed in $O(h)$ time. By using this parameter in a similar analysis than in Theorem 5.1 (with the potential of a node $\Phi(z) = h(z)$), we can show that our self-adjusting skip list satisfies the bound $\frac{WS(X)}{\lg B}$.

If we represent skip lists as B-trees using the detailed reduction of Dean and Jones [14], we obtain a self-adjusting B-tree which is dynamically optimal in the external-memory model.

References

- [1] G. Adel'so-Vel'skii and E. Landis. An algorithm for the organisation of information. *Dokl. Akad. Nauk SSSR* 146 (1962), 263-266 (in Russian); English Translation in *Soviet. Math.* 3, 1259-1262.

- [2] A. Aggarwal and J. Vitter. The input/output complexity of sorting and related problems. *CACM*, 31(9):1116–1127, 1988.
- [3] S. Albers. Improved randomized on-line algorithms for the list update problem. In *Siam Journal on Computing*, volume 11(3), pages 682–693, 1998.
- [4] C. Ambühl. Offline list update is np-hard. In *Proceedings of the 8th Annual European Symposium (ESA 2000)*, volume 1879 of LNCS, pages 42–51, 2000.
- [5] C. R. Aragon and R. Seidel. Randomized search trees. In *Algorithmica*, volume 16, pages 464–497, 1996.
- [6] A. Bagchi, A. L. Buchsbaum, and M. T. Goodrich. Biased skip lists. In *Algorithmica*, volume 42(1), pages 31–48, 2005.
- [7] R. Bayer. Symmetric binary b-trees: Data structure and maintenance algorithms. In *Acta Informatica*, volume 1, pages 290–306, 1972.
- [8] R. Bayer and E. McCreight. Organization and maintenance of large ordered indexes. In *Acta Informatica*, volume 1, pages 173–189, 1972.
- [9] S. W. Bent, D. Sleator, and R. Tarjan. Biased search trees. In *SIAM Journal on Computing*, volume 14(3), pages 545–568, 1985.
- [10] M. Bădoiu, R. Cole, E. D. Demaine, and J. Iacono. A unified access bound on comparison-based dynamic dictionaries. *Theoretical Computer Science*, 382(2):86–96, 2007.
- [11] V. Ciriani, P. Ferragina, F. Luccio, and S. Muthukrishnan. A data structure for a sequence of string accesses in external memory. In *ACM Transactions on Algorithms*, volume 3(1), 2007.
- [12] R. Cole. Part II: The proof. In *Siam J. Comput.*, volume 30, pages 44–85, 2000.
- [13] R. Cole, B. Mishra, J. Schmidt, and A. Siegel. On the dynamic finger conjecture for splay trees. Part I: Splay sorting $\log n$ -block sequences. In *Siam J. Comput.*, volume 30, pages 1–43, 2000.
- [14] B. C. Dean and Z. H. Jones. Exploring the duality between skip lists and binary search trees. In *Proceedings of the 45th annual southeast regional conference*, pages 395–399, 2007.
- [15] E. Demaine, D. Harmon, J. Iacono, and M. Pătrașcu. Dynamic optimality—almost. In *SIAM journal on Computing*, to appear. *Special issue of selected papers from the 45th Annual IEEE Symposium on Foundations of Computer Science*.
- [16] J. Derryberry, D. D. Sleator, and C. C. Wang. A lower bound framework for binary search trees with rotations. Technical report, Computer Science Department, School of Computer Science, Carnegie Mellon University, 2005.
- [17] J. Derryberry, D. D. Sleator, and C. C. Wang. $O(\log \log n)$ -competitive dynamic binary search trees. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm (SODA '06)*, pages 374–383, 2006.
- [18] A. Elmasry. On the sequential access theorem and deque conjecture for splay trees. In *Theoretical Computer Science*, volume 314, pages 459–466, 2004.
- [19] F. Ergun, S. Mittra, S. C. Sahinalp, J. Sharp, and R. Sinha. A dynamic lookup scheme for bursty access patterns. In *Proceedings of IEEE INFOCOM*, 2001.
- [20] F. Ergun, S. C. Sahinalp, J. Sharp, and R. Sinha. Biased dictionaries with fast insert/deletes. In *Proceedings of the 33rd annual ACM Symposium on Theory of Computing (STOC)*, pages 483–491, 2001.
- [21] F. Ergun, S. C. Sahinalp, J. Sharp, and R. Sinha. Biased skip lists for highly skewed access patterns. In *Proceedings of ALENEX*, 2001.
- [22] J. Feigenbaum and R. Tarjan. Two new kinds of biased search trees. In *Bell System Technical Journal*, volume 62(10), pages 3139–3158, 1983.
- [23] G. F. Georgakopoulos. Splay trees: a reweighing lemma and a proof of competitiveness vs. dynamic balanced trees. In *Journal of Algorithms*, volume 51(1), pages 64–76, 2004.
- [24] G. F. Georgakopoulos. How to splay for loglog n -competitiveness. In *Workshop on Experimental and Efficient Algorithms*, pages 570–579, 2005.
- [25] S. Huddleston and K. Mehlhorn. A new data structure for representing sorted lists. In *Acta Informatica*, volume 17, pages 157–184. Springer, 1982.
- [26] J. Iacono. Alternatives to splay trees with $O(\log n)$ worst-case access times. In *Proceedings of the twelfth annual ACM-SIAM Symposium on Discrete Algorithms*, pages 516–522, 2001.
- [27] R. Karp and P. Raghavan. From a personal communication cited in [36].
- [28] R. S. Leonidas J. Guibas. A dichromatic framework for balanced trees. In *Proc. 19th IEEE Symp. on Foundations of Computer Science*, pages 8–21, 1978.
- [29] C. U. Martel. Self-adjusting multi-way search trees. *Inf. Process. Lett.*, 38(3):135–141, 1991.
- [30] C. Martinez and S. Roura. Optimal and nearly optimal static weighted skip lists. Technical report, LSI-95-34-R, Dept. Llenguatges i Sistemes Informàtics (Universitat Politècnica de Catalunya), 1995.
- [31] K. Mehlhorn and S. Näher. Algorithm design and software libraries: Recent developments in the leda project. In *IFIP 12th World Computer Congress*, volume 1, pages 493–505, 1992.
- [32] X. Messeguer. Skip trees, an alternative data structure to skip lists in a concurrent approach. *Informatique Theorique et Applications*, 31(3):251–269, 1997.
- [33] K. Mulmuley. *Computational Geometry: An Introduction Through Randomized Algorithm, exercise 1.4.8*. Prentice-Hall, 1994.
- [34] I. Munro, T. Papadakis, and R. Sedgewick. Deterministic skip lists. In *Proceedings of the third annual ACM-SIAM symposium on Discrete algorithms*, pages 367–375, 1992.
- [35] W. Pugh. Skip lists: a probabilistic alternative to balanced trees. In *Communications of the ACM*, volume 33(6), pages 668–676, 1990.
- [36] N. Reingold, J. Westbrook, and D. D. Sleator. Randomized competitive algorithms for the list update

- problem. In *Algorithmica*, volume 11(1), pages 15–32, 1994.
- [37] M. Sherk. Self-adjusting k-ary search trees. In *Proceedings of the Workshop on Algorithms and Data Structures*, volume 382, pages 381–392, 1989.
- [38] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. In *Communications of the ACM*, volume 28(2), pages 202–208, 1985.
- [39] D. D. Sleator and R. E. Tarjan. Self-adjusting binary search trees. *J. ACM*, 32(3):652–686, 1985.
- [40] R. Sundar. Sciencetwists, turns, cascades, deque conjecture, and scanning theorem. In *Proceedings of the 13th Symposium on Foundations of Computer*, pages 555–559, 1989.
- [41] R. Sundar. On the deque conjecture for the splay algorithm. In *Combinatorica*, volume 12, pages 95–124, 1992.
- [42] R. Tarjan. Sequential access in splay trees takes linear time. In *Combinatorica*, volume 5, pages 367–378, 1985.
- [43] B. Teia. A lower bound for randomized list update algorithms. In *Information Processing Letters*, volume 47(1), pages 5–9, 1993.
- [44] F. L. V. Ciriani, P. Ferragina and S. Muthukrishnan. Self-adjusting data structures for external memory string access. Technical report, TR-01-26 (Università Di Pisa), 2001.
- [45] R. Wilber. Lower bounds for accessing binary search trees with rotations. In *SIAM journal on Computing*, volume 18(1), pages 56–67, 1989.