Self-Adjusting Binary Search Trees: What Makes Them Tick?

Parinya Chalermsook[1], Mayank Goswami[1], László Kozma[2], Kurt Mehlhorn[1], and Thatchaphol Saranurak[3,⋆]

[1] Max-Planck Institute for Informatics, Saarbrücken, 66123, Germany
[2] Department of Computer Science, Saarland University, Saarbrücken, 66123, Germany
[3] KTH Royal Institute of Technology, Stockholm, 11428, Sweden

Abstract. Splay trees (Sleator and Tarjan [11]) satisfy the so-called *access lemma*. Many of the nice properties of splay trees follow from it. *What makes self-adjusting binary search trees (BSTs) satisfy the access lemma?* After each access, self-adjusting BSTs replace the search path by a tree on the same set of nodes (the after-tree). We identify two simple combinatorial properties of the search path and the after-tree that imply the access lemma. Our main result

(i) implies the access lemma for *all* minimally self-adjusting BST algorithms for which it was known to hold: splay trees and their generalization to the class of *local algorithms* (Subramanian [12], Georgakopoulos and McClurkin [7]), as well as Greedy BST, introduced by Demaine et al. [5] and shown to satisfy the access lemma by Fox [6],
(ii) implies that BST algorithms based on "strict" depth-halving satisfy the access lemma, addressing an open question that was raised several times since 1985, and
(iii) yields an extremely short proof for the $O(\log n \log\log n)$ amortized access cost for the path-balance heuristic (proposed by Sleator), matching the best known bound (Balasubramanian and Raman [2]) to a lower-order factor.

One of our combinatorial properties is *locality*. We show that any BST-algorithm that satisfies the access lemma via the sum-of-log (SOL) potential is necessarily local. The other property states that the sum of the number of leaves of the after-tree plus the number of side alternations in the search path must be at least a constant fraction of the length of the search path. We show that a weak form of this property is necessary for sequential access to be linear.

1 Introduction

The binary search tree (BST) is a fundamental data structure for the dictionary problem. Self-adjusting BSTs rearrange the tree in response to data accesses, and are thus able to adapt to the distribution of queries. We consider the class of *minimally self-adjusting* BSTs: algorithms that rearrange only the search path during each access and make the accessed element the root of the tree. Let s be the element accessed and let P be the search path to s. Such an algorithm can be seen as a mapping from the search path P

⋆ Work done while at Saarland University.

N. Bansal and I. Finocchi (Eds.): ESA 2015, LNCS 9294, pp. 300–312, 2015.
DOI: 10.1007/978-3-662-48350-3_26

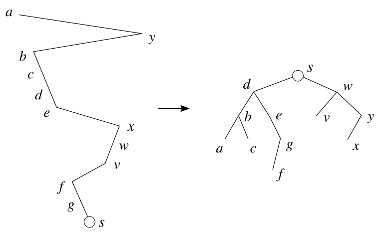

Fig. 1. The search path to s is shown on the left, and the after-tree is shown on the right. The search path consists of 12 nodes and contains four edges that connect nodes on different sides of s ($z = 4$ in the language of Theorem 1). The after-tree has five leaves. The left-depth of a in the after-tree is three (the path from the root to a goes left three times) and the right-depth of y is two. The set $\{a, c, f, v, y\}$ is subtree-disjoint. The sets $\{d, e, g\}$, $\{b, f\}$, $\{x, y\}$, $\{w\}$ are monotone.

(called "before-path" in the sequel) to a tree A with root s on the same set of nodes (called "after-tree" in the sequel). Observe that all subtrees that are disjoint from the before-path can be reattached to the after-tree in a unique way governed by the ordering of the elements. In the BST model, the cost of the access plus the cost of rearranging is $|P|$, see Figure 1 for an example.

Let T be a binary search tree on $[n]$. Let $w : [n] \to \mathbb{R}_{>0}$ be a positive weight function, and for any set $S \subseteq [n]$, let $w(S) = \sum_{a \in S} w(a)$. Sleator and Tarjan defined the sum-of-log (SOL) potential function $\Phi_T = \sum_{a \in [n]} \log w(T_a)$, where T_a is the subtree of T rooted at a. We say that an algorithm $\mathcal{A}$ satisfies the *access lemma (via the SOL potential function)* if for all T' that can be obtained as a rearrangement done by algorithm $\mathcal{A}$ after some element s is accessed, we have

$$|P| \leq \Phi_T - \Phi_{T'} + O(1 + \log \frac{W}{w(s)}),$$

where P is the search path when accessing s in T and $W = w(T)$. The access lemma is known to hold for the splay trees of Sleator and Tarjan [11], for their generalizations to *local algorithms* by Subramanian [12] and Georgakopoulos and McClurkin [7], as well as for Greedy BST, an online algorithm introduced by Demaine et al. [5] and shown to satisfy the access lemma by Fox [6]. For minimally self-adjusting BSTs, the access lemma implies *logarithmic amortized cost*, *static optimality*, and the *static finger* and *working set* properties.

Theorem 1. *Let $\mathcal{A}$ be a minimally self-adjusting BST algorithm. If (i) the number of leaves of the after-tree is $\Omega(|P| - z)$ where P is the search path and z is the number of "side alternations*[1]*" in P and (ii) for any element $t > s$ (resp. $t < s$), the right-depth of t (left-depth of t) in the after-tree is $O(1)$, then $\mathcal{A}$ satisfies the access lemma.*

Note that the conditions in Theorem 1 are purely combinatorial conditions on the before-paths and after-trees. In particular, the potential function is completely hidden. The theorem directly implies the access lemma for all BST algorithms mentioned above and some new ones.

Corollary 2. *The following BST algorithms satisfy the access lemma: (i) Splay tree, as well as its generalizations to local algorithms (ii) Greedy BST, and (iii) new heuristics based on "strict" depth-halving.*

[1] z is the number of edges on the search path connecting nodes on different sides of s. The right-depth of a node is the number of right-going edges on the path from the root to the node.

The third part of the corollary addresses an open question raised by several authors [12,2,7] about whether some form of depth reduction is sufficient to guarantee the access lemma. We show that a strict depth-halving suffices.

For the first part, we formulate a global view of splay trees. We find this new description intuitive and of independent interest. The proof of (i) is only a few lines.

We also prove a partial converse of Theorem 1.

Theorem 3 (Partial Converse). *If a BST algorithm satisfies the access lemma via the SOL-potential function, the after-trees must satisfy condition (ii) of Theorem 1.*

We call a BST algorithm *local* if the transformation from before-path to after-tree can be performed in a bottom-up traversal of the path with a buffer of constant size. Nodes outside the buffer are already arranged into subtrees of the after-tree. We use Theorem 3 to show that BST-algorithms satisfying the access lemma (via the SOL-potential) are necessarily local.

Theorem 4 (Characterization Theorem). *If a minimally self-adjusting BST algorithm satisfies the access lemma via the SOL-potential, then it is local.*

The theorem clarifies, why the access lemma was shown only for local BST algorithms.

In the following, we introduce our main technical tools: subtree-disjoint and monotone sets in § 2, and zigzag sets in § 3. Bounding the potential change over these sets leads to the proof of Theorem 1 in § 3. Corollary 2(i) is also proved in § 3. Corollary 2(iii) is the subject of § 4. In § 5.1 we show that condition (ii) of Theorem 1 is necessary (Theorem 3), and in § 5.2 we argue that a weaker form of condition (i) must also be fulfilled by any reasonably efficient algorithm. We prove Theorem 4 in § 6. For lack of space, in this version of the paper we omit some of the proofs (e.g. Corollary 2(ii)). We refer the reader to the preprint version [3] for full details.

Notation: We use T_a or $T(a)$ to denote the subtree of T rooted at a. We use the same notation to denote the set of elements stored in the subtree. The set of elements stored in a subtree is an interval of elements. If c and d are the smallest and largest elements in $T(a)$, we write $T(a) = [c, d]$. We also use open and half-open intervals to denote subsets of $[n]$, for example $[3, 7)$ is equal to $\{3, 4, 5, 6\}$. We frequently write Φ instead of Φ_T and Φ' instead of $\Phi_{T'}$.

2 Disjoint and Monotone Sets

Let $\mathcal{A}$ be any BST algorithm. Consider an access to s and let T and T' be the search trees before and after the access. The main task in proving the access lemma is to relate the potential difference $\Phi_T - \Phi_{T'}$ to the length of the search path. For our arguments, it is convenient to split the potential into parts that we can argue about separately. For a subset X of the nodes, define a partial potential on X as $\Phi_T(X) = \sum_{a \in X} \log w(T(a))$.

We start with the observation that the potential change is determined only by the nodes on the search path and that we can argue about disjoint sets of nodes separately.

Proposition 5. *Let P be the search path to s. For $a \notin P$, $T(a) = T'(a)$. Therefore, $\Phi_T - \Phi_{T'} = \Phi_T(P) - \Phi_{T'}(P)$. Let $X = \dot{\bigcup}_{i=1}^{k} X_i$ where the sets X_i are pairwise disjoint. Then $\Phi_T(X) - \Phi_{T'}(X) = \sum_{i=1}^{k}(\Phi_T(X_i) - \Phi_{T'}(X_i))$.*

We introduce three kinds of sets of nodes, namely subtree-disjoint, monotone, and zigzag sets, and derive bounds for the potential change for each one of them. A subset X of the search path is *subtree-disjoint* if $T'(a) \cap T'(a') = \emptyset$ for all pairs $a \neq a' \in X$; remark that subtree-disjointness is defined w.r.t. the subtrees after the access. We bound the change of partial potential for subtree-disjoint sets. The proof of the following lemma was inspired by the proof of the access lemma for Greedy BST by Fox [6].

Lemma 6. *Let X be a subtree-disjoint set of nodes. Then*

$$|X| \le 2 + 8 \cdot \log \frac{W}{w(T(s))} + \Phi_T(X) - \Phi_{T'}(X).$$

Proof: We consider the nodes smaller than s and greater or equal to s separately, i.e. $X = X_{<s} \dot{\cup} X_{\ge s}$. We show $|X_{\ge s}| \le 1 + \Phi_T(X_{\ge s}) - \Phi_{T'}(X_{\ge s}) + 4 \log \frac{W}{w(T(s))}$, and the same holds for $X_{<s}$. We only give the proof for $X_{\ge s}$.

Denote $X_{\ge s}$ by $Y = \{a_0, a_1, \ldots, a_q\}$ where $s \le a_0 < \ldots < a_q$. Before the access, s is a descendant of a_0, a_0 is a descendant of a_1, and so on. Let $T(a_0) = [c, d]$. Then $[s, a_0] \subseteq [c, d]$ and $d < a_1$. Let $w_0 = w(T(a_0))$. For $j \ge 0$, define σ_j as the largest index ℓ such that $w([c, a_\ell]) \le 2^j w_0$. Then $\sigma_0 = 0$ since weights are positive and $[c, d]$ is a proper subset of $[c, a_1]$. The set $\{\sigma_0, \ldots\}$ contains at most $\lceil \log(W/w_0) \rceil$ distinct elements. It contains 0 and q.

Now we upper bound the number of i with $\sigma_j \le i < \sigma_{j+1}$. We call such an element a_i *heavy* if $w(T'(a_i)) > 2^{j-1} w_0$. There can be at most 3 heavy elements as otherwise $w([c, a_{j+1}]) \ge \sum_{\sigma_j \le k < \sigma_{j+1}} w(T'(a_k)) > 4 \cdot 2^{j-1} w_0$, a contradiction.

Next we count the number of light (= non-heavy) elements. For each such light element a_i, we have $w(T'(a_i)) \le 2^{j-1} w_0$. We also have $w(T(a_{i+1})) \ge w([c, a_{i+1}]) > w([c, a_{\sigma_j}])$ and thus $w(T(a_{i+1})) > 2^j w_0$ by the definition of σ_j. Thus the ratio $r_i = w(T(a_{i+1}))/w(T'(a_i)) \ge 2$ whenever a_i is a light element. Moreover, for any $i = 0, \ldots, q-1$ (for which a_i is not necessarily light), we have $r_i \ge 1$. Thus,

$$2^{\text{number of light elements}} \le \prod_{0 \le i \le q-1} r_i = \left(\prod_{0 \le i \le q} \frac{w(T(a_i))}{w(T'(a_i))} \right) \cdot \frac{w(T'(a_q))}{w_0}.$$

So the number of light elements is at most $\Phi_T(Y) - \Phi_{T'}(Y) + \log(W/w_0)$.
Putting the bounds together, we obtain, writing L for $\log(W/w_0)$:

$$|Y| \le 1 + 3(\lceil L \rceil - 1) + \Phi_T(Y) - \Phi_{T'}(Y) + L \le 1 + 4L + \Phi_T(Y) - \Phi_{T'}(Y).$$

□

Now we proceed to analyze our second type of subsets, that we call *monotone sets.* A subset X of the search path is *monotone* if all elements in X are larger (smaller) than s and have the same right-depth (left-depth) in the after-tree.

Lemma 7. *Assume $s < a < b$ and that a is a proper descendant of b in P. If $\{a, b\}$ is monotone, $T'(a) \subseteq T(b)$.*

Proof: Clearly $[s, b] \subseteq T(b)$. The smallest item in $T'(a)$ is larger than s, and, since a and b have the same right-depth, b is larger than all elements in $T'(a)$. □

Lemma 8. *Let X be a monotone set of nodes. Then*

$$\Phi(X) - \Phi'(X) + \log \frac{W}{w(s)} \geq 0.$$

Proof: We order the elements in $X = \{a_1, \ldots, a_q\}$ such that a_i is a proper descendant of a_{i+1} in the search path for all i. Then $T'(a_i) \subseteq T(a_{i+1})$ by monotonicity, and hence

$$\Phi(X) - \Phi'(X) = \log \frac{\prod_{a \in X} w(T(a))}{\prod_{a \in X} w(T'(a))} = \log \frac{w(T(a_1))}{w(T'(a_q))} + \sum_{i=1}^{q-1} \log \frac{w(T(a_{i+1}))}{w(T'(a_i))}.$$

The second sum is nonnegative. Thus $\Phi(X) - \Phi'(X) \geq \log \frac{w(T(a_1))}{w(T'(a_q))} \geq \log \frac{w(s)}{W}$. □

Theorem 9. *Suppose that, for every access to an element s, we can partition the elements on the search path P into at most k subtree-disjoint sets D_1 to D_k and at most ℓ monotone sets M_1 to M_ℓ. Then*

$$\sum_{i \leq k} |D_i| \leq \Phi_T(S) - \Phi_{T'}(S) + 2k + (8k + \ell) \log \frac{W}{w(s)}.$$

The proof of Theorem 9 follows immediately from Lemma 6 and 8. We next give some easy applications.

Path-Balance: The path-balance algorithm maps the search path P to a balanced BST of depth $c = \lceil \log_2(1 + |P|) \rceil$ rooted at s. Then

Lemma 10. $|P| \leq \Phi(P) - \Phi'(P) + O((1 + \log |P|)(1 + \log(W/w(s))))$.

Proof: We decompose P into sets P_0 to P_c, where P_k contains the nodes of depth k in the after-tree. Each P_k is subtree-disjoint. An application of Theorem 9 completes the proof. □

Theorem 11. *Path-Balance has amortized cost at most $O(\log n \log \log n)$.*

Proof: We choose the uniform weight function: $w(a) = 1$ for all a. Let c_i be the cost of the i-th access, $1 \leq i \leq m$, and let $C = \sum_{1 \leq i \leq m} c_i$ be the total cost of the accesses. Note that $\prod_i c_i \leq (C/m)^m$. The potential of a tree with n items is at most $n \log n$. Thus $C \leq n \log n + \sum_{1 \leq i \leq m} O((1 + \log c_i)(1 + \log n)) = O((n + m) \log n) + O(m \log n) \cdot \log(C/m)$ by Lemma 10. Assume $C = K(n + m) \log n$ for some K. Then $K = O(1) + O(1) \cdot \log(K \log n)$ and hence $K = O(\log \log n)$. □

Greedy BST: The Greedy BST algorithm was introduced by Demaine et al. [5]. It is an online version of the offline greedy algorithm proposed independently by Lucas and Munro [9,8]. The definition of Greedy BST requires a geometric view of BSTs. Our notions of subtree-disjoint and monotone sets translate naturally into geometry, and this allows us to derive the following theorem.

Theorem 12. *Greedy BST satisfies the (geometric) access lemma.*

The geometric view of BSTs and the proof of Theorem 12 are omitted here. We refer to the preprint version [3] for details. We remark that once the correspondences to geometric view are defined, the proof of Theorem 12 is almost immediate.

3 Zigzag Sets

Let s be the accessed element and let $a_1, \ldots, a_{|P|-1}$ be the reversed search path without s. For each i, define the set $Z_i = \{a_i, a_{i+1}\}$ if a_i and a_{i+1} lie on different sides of s, and let $Z_i = \emptyset$ otherwise. The zigzag set Z_P is defined as $Z_P = \bigcup_i Z_i$. In words, the number of non-empty sets Z_i is exactly the number of "side alternations" in the search path, and the cardinality of Z_P is the number of elements involved in such alternations.

Rotate to Root: We first analyze the rotate-to-root algorithm (Allen, Munro [1]), that brings the accessed element s to the root and arranges the elements smaller (larger) than s so the ancestor relationship is maintained, see Figure 2 for an illustration.

Lemma 13. $|Z| \leq \Phi(Z_P) - \Phi'(Z_P) + O(1 + \log \frac{W}{w(T(s))})$.

Proof: Because s is made the root and ancestor relationships are preserved otherwise, $T'(a) = T(a) \cap (-\infty, s)$ if $a < s$ and $T'(a) = T(a) \cap (s, \infty)$ if $a > s$. We first deal with a single side alternation.

Claim. $2 \leq \Phi(Z_i) - \Phi'(Z_i) + \log \frac{w(T(a_{i+1}))}{w(T(a_i))}$.

Proof: This proof is essentially the proof of the zig-zag step for splay trees. We give the proof for the case where $a_i > s$ and $a_{i+1} < s$; the other case is symmetric. Let a' be the left ancestor of a_{i+1} in P and let a'' be the right ancestor of a_i in P. If these elements do not exist, they are $-\infty$ and $+\infty$, respectively. Let $W_1 = w((a', 0))$, $W_2 = w((0, a''))$, and $W' = w((a_{i+1}, 0))$. In T, we have $w(T(a_i)) = W' + w(s) + W_2$ and $w(T(a_{i+1})) = W_1 + w(s) + W_2$, and in T', we have $w(T'(a_i)) = W_2$ and $w(T'(a_{i+1})) = W_1$.

Thus $\Phi(Z_i) - \Phi'(Z_i) + \log \frac{W_1 + w(s) + W_2}{W' + w(s) + W_2} \geq \log(W_1 + w(s) + W_2) - \log W_1 + \log(W_2 + w(s) + W') - \log W_2 + \log \frac{W_1 + w(s) + W_2}{W' + w(s) + W_2} \geq 2\log(W_1 + W_2) - \log W_1 - \log W_2 \geq 2$, since $(W_1 + W_2)^2 \geq 4W_1W_2$ for all positive numbers W_1 and W_2. $\square$

Let Z_{even} (Z_{odd}) be the union of the Z_i with even (odd) indices. One of the two sets has cardinality at least $|Z_P|/2$. Assume that it is the former; the other case is symmetric. We sum the statement of the claim over all i in Z_{even} and obtain

$$\sum_{i \in Z_{\text{even}}} \left(\Phi(Z_i) - \Phi'(Z_i) + \log \frac{w(T(a_{i+1}))}{w(T(a_i))} \right) \geq 2\,|Z_{\text{even}}| \geq |Z_P|\,.$$

The elements in $Z_P \setminus Z_{\text{even}}$ form two monotone sets and hence $\Phi(Z_P \setminus Z_{even}) - \Phi'(Z_P \setminus Z_{even}) + 2\log(W/w(s)) \geq 0$. This completes the proof. □

The following theorem combines all three tools we have introduced: subtree-disjoint, monotone, and zigzag sets.

Theorem 14. *Suppose that, for every access we can partition $P \setminus s$ into at most k subtree-disjoint sets D_1 to D_k and at most ℓ monotone sets M_1 to M_ℓ. Then*

$$\sum_{i \leq k} |D_i| + |Z_P| \leq \Phi(P) - \Phi'(P) + O((k+\ell)(1 + \log \frac{W}{w(s)})).$$

Proof: We view the transformation as a two-step process, i.e., we first rotate s to the root and then transform the left and right subtrees of s. Let Φ'' be the potential of the intermediate tree. By Lemma 13, $|Z_P| \leq \Phi(P) - \Phi''(P) + O(1 + \log \frac{W}{w(T(s))})$. By Theorem 9, $\sum_{i \leq k} |D_i| \leq \Phi''(P) - \Phi'(P) + O((k+\ell)(1 + \log \frac{W}{w(T(s))}))$. □

We next derive an easy to apply corollary from this theorem. For the statement, we need the following proposition that follows directly from the definition of monotone set.

Proposition 15. *Let S be a subset of the search path consisting only of elements larger than s. Then S can be decomposed into ℓ monotone sets if and only if the elements of S have only ℓ different right-depths in the after-tree.*

Theorem 16 (Restatement of Theorem 1). *Suppose the BST algorithm $\mathcal{A}$ rearranges a search path P that contains z side alternations, into a tree A such that (i) s, the element accessed, is the root of A, (ii) the number of leaves of A is $\Omega(|P| - z)$, (iii) for every element x larger (smaller) than s, the right-depth (left-depth) of x in A is bounded by a constant. Then $\mathcal{A}$ satisfies the access lemma.*

Proof: Let B be the set of leaves of T and let $b = |B|$. By assumption (ii), there is a positive constant c such that $b \geq (|T| - z)/c$. Then $|T| \leq cb + z$. We decompose $P \setminus s$ into B and ℓ monotone sets. By assumption (iii), $\ell = O(1)$. An application of Theorem 14 with $k = 1$ and $\ell = O(1)$ completes the proof. □

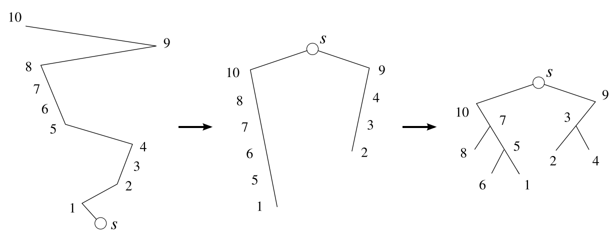

Fig. 2. A global view of splay trees. The transformation from the left to the middle illustrates rotate-to-root. The transformation from the left to the right illustrates splay trees.

Splay: Splay extends rotate-to-root: Let $s = v_0, v_1, \ldots v_k$ be the reversed search path. We view splaying as a two step process, see Figure 2. We first make s the root and split the search path into two paths, the path of elements smaller than s and the path of elements larger than s. If v_{2i+1} and v_{2i+2} are on the same side of s, we rotate them, i.e., we remove v_{2i+2} from the path and make it a child of v_{2i+1}.

Proposition 17. *The above description of splay is equivalent to the Sleator-Tarjan description.*

Theorem 18. *Splay satisfies the access lemma.*

Proof: There are $|P|/2 - 1$ odd-even pairs. For each pair, if there is no side change, then splay creates a new leaf in the after-tree. Thus

$$\text{\# of leaves} \geq |P|/2 - 1 - \text{\# of side changes}.$$

Since right-depth (left-depth) of elements in the after-tree of splay is at most 2, an application of Theorem 16 finishes the proof. □

4 New Heuristics: Depth Reduction

Already Sleator and Tarjan [11] formulated the belief that *depth-halving* is the property that makes splaying efficient, i.e. the fact that every element on the access path reduces its distance to the root by a factor of approximately two. Later authors [12,2,7] raised the question, whether a suitable *global* depth-reduction property is sufficient to guarantee the access lemma. Based on Theorem 16, we show that a strict form of depth-halving suffices to guarantee the access lemma.

Let x and y be two arbitrary nodes on the search path. If y is an ancestor of x in the search path, but not in the after-tree, then we say that x has *lost* the ancestor y, and y has lost the descendant x. Similarly we define *gaining* an ancestor or a descendant. We stress that only nodes on the search path (resp. the after-tree) are counted as descendants, and not the nodes of the pendent trees. Let $d(x)$ denote the depth (number of ancestors) of x in the search path. We give a sufficient condition for a good heuristic, stated below. The proof is omitted.

Theorem 19. *Let $\mathcal{A}$ be a minimally self-adjusting BST algorithm that satisfies the following conditions: (i) Every node x on the search path loses at least $(\frac{1}{2} + \epsilon) \cdot d(x) - c$ ancestors, for fixed constants $\epsilon > 0, c > 0$, and (ii) every node on the search path, except the accessed element, gains at most d new descendants, for a fixed constant $d > 0$. Then $\mathcal{A}$ satisfies the access lemma.*

We remark that in general, splay trees do not satisfy condition (i) of Theorem 19. One may ask how tight are the conditions of Theorem 19. If we relax the constant in condition (i) from $(\frac{1}{2} + \epsilon)$ to $\frac{1}{2}$, the conditions of Theorem 16 are no longer implied. There exist rearrangements in which every node loses a $\frac{1}{2}$-fraction of its ancestors, gains at most two ancestors or descendants, yet both the number of side alternations and the number of leaves created are $O(\sqrt{|P|})$, where P is the before-path (details can be found in [3]). If we further relax the ratio to $(\frac{1}{2} - \epsilon)$, we can construct an example where the number of alternations and the number of leaves created are only $O(\log |P|/\epsilon)$.

Allowing more gained descendants and limiting instead the number of gained ancestors is also beyond the strength of Theorem 16. It is possible to construct an example [3] in which every node loses an $(1 - o(1))$-fraction of ancestors, yet the number of leaves created is only $O(\sqrt{|P|})$ (while having no alternations in the before-path).

Finally, we observe that depth-reduction alone is likely not sufficient: one can restructure the access path in such a way that every node reduces its depth by a constant factor, yet the resulting after-tree has an anti-monotone path of linear size [3]. Based on Theorem 20, this means that if such a restructuring were to satisfy the access lemma in its full generality, the SOL potential would not be able to show it.

5 Necessary Conditions

5.1 Necessity of $O(1)$ Monotone Sets

In this section we show that condition (ii) of Theorem 1 is necessary for any minimally self-adjusting BST algorithm that satisfies the access lemma via the SOL potential function.

Theorem 20. *Consider the transformations from before-path P to after-tree A by algorithm $\mathcal{A}$. If $A \setminus s$ cannot be decomposed into constantly many monotone sets, then $\mathcal{A}$ does not satisfy the access lemma with the SOL potential.*

Proof: We may assume that the right subtree of A cannot be decomposed into constantly many monotone sets. Let $x > s$ be a node of maximum right depth in A. By Lemma 15, we may assume that the right depth is $k = \omega(1)$. Let $a_{i_1}, \ldots, a_{i_k}$ be the elements on the path to x where the right child pointer is used. All these nodes are descendants of x in the before-path P.

We now define a weight assignment to the elements of P and the pendent trees for which the access lemma does not hold with the SOL potential. We assign weight zero to all pendent trees, weight one to all proper descendants of x in P and weight K to all ancestors of x in P. Here K is a big number. The total weight W then lies between K and $|P|\,K$.

We next bound the potential change. Let $r(a_i) = w(T'(a_i))/w(T(a_i))$ be the ratio of the weight of the subtree rooted at a_i in the after-tree and in the before-path. For any element a_{i_j} at which a right turn occurs, we have $w(T(a_{i_j})) \leq |P|$ and $w(T'(a_{i_j})) \geq K$. So $r(a_{i_j}) \geq K/|P|$. Consider now any other a_i. If it is an ancestor of x in the before-path, then $w(T(a_i)) \leq W$ and $w(T'(a_i)) \geq K$. If it is a descendant of x, then $w(T(a_i)) \leq |P|$ and $w(T'(a_i)) \geq 1$. Thus $r(a_i) \geq 1/\,|P|$ for every a_i. We conclude

$$\Phi'(T) - \Phi(T) \geq k \cdot \log \frac{K}{|P|} - |P| \log |P|.$$

If $\mathcal{A}$ satisfies the access lemma with the SOL potential function, then we must have $\Phi'(T) - \Phi(T) \leq O(\log \frac{W}{w(s)} - |P|) = O(\log(K\,|P|))$. However, if K is large enough and $k = \omega(1)$, then $k \cdot \lg \frac{K}{|P|} - |P| \lg |P| \gg O(\log(K\,|P|))$. □

5.2 Necessity of Many Leaves

In this section we study condition (i) of Theorem 1. We show that some such condition is necessary for an efficient BST algorithm: if a local algorithm consistently creates only few leaves, it cannot satisfy the sequential access theorem, a natural efficiency condition known to hold for several BST algorithms [13,6].

Definition 21. *A self-adjusting BST algorithm $\mathcal{A}$ satisfies the* sequential access theorem *if starting from an arbitrary initial tree T, it can access the elements of T in increasing order with total cost $O(|T|)$.*

Theorem 22. *If for all after-trees A created by algorithm $\mathcal{A}$ executed on T, it holds that (i) A can be decomposed into $O(1)$ monotone sets, and (ii) the number of leaves of A is at most $|T|^{o(1)}$, then $\mathcal{A}$ does not satisfy the sequential access theorem.*

The rest of the section is devoted to the proof of Theorem 22.

Let R be a BST over $[n]$. We call a maximal left-leaning path of R a *wing* of R. More precisely, a wing of R is a set $\{x_1, \ldots, x_k\} \subseteq [n]$, with $x_1 < \cdots < x_k$, and such that x_1 has no left child, x_k is either the root of R, or the right child of its parent, and x_i is the left child of x_{i+1} for all $1 \le i < k$. A wing might consist of a single element. Observe that the wings of R partition $[n]$ in a unique way, and we call the set of wings of R the *wing partition* of R, denoted as $wp(R)$. We define a potential function ϕ over a BST R as follows: $\phi(R) = \sum_{w \in wp(R)} |w| \log(|w|)$.

Let T_0 be a left-leaning path over $[n]$ (i.e. n is the root and 1 is the leaf). Consider a minimally self-adjusting BST algorithm $\mathcal{A}$, accessing elements of $[n]$ in sequential order, starting with T_0 as initial tree. Let T_i denote the BST after accessing element i. Then T_i has i as the root, and the elements yet to be accessed (i.e. $[i+1, n]$) form the right subtree of the root, denoted R_i. To avoid treating T_0 separately, we augment it with a "virtual root" 0. This node plays no role in subsequent accesses, and it only adds a constant one to the overall access cost.

Using the previously defined potential function, we denote $\phi_i = \phi(R_i)$. We make the following easy observations: $\phi_0 = n \log n$, and $\phi_n = 0$.

Next, we look at the change in potential due to the restructuring after accessing element i. Let $P_i = (x_1, x_2, \ldots, x_{n_i})$ be the access path when accessing i in T_{i-1}, and let n_i denote its length, i.e. $x_1 = i-1$, and $x_{n_i} = i$. Observe that the set $P_i' = P_i \setminus \{x_1\}$, is a wing of T_{i-1}.

Let us denote the after-tree resulting from rearranging the path P_i as A_i. Observe that the root of A_i is i, and the left child of i in A_i is $i-1$. We denote the tree $A_i \setminus \{i-1\}$ as A_i', and the tree $A_i' \setminus \{i\}$, i.e. the right subtree of i in A_i, as A_i''.

The crucial observation of the proof is that for an arbitrary wing $w \in wp(T_i)$, the following holds: (i) either w was not changed when accessing i, i.e. $w \in wp(T_{i-1})$, or (ii) w contains a portion of P_i', possibly concatenated with an earlier wing, i.e. there exists some $w' \in wp(A_i')$, such that $w' \subseteq w$. In this case, we denote $\mathrm{ext}(w')$ the *extension* of w' to a wing of $wp(T_i)$, i.e. $\mathrm{ext}(w') = w \setminus w'$, and either $\mathrm{ext}(w') = \emptyset$, or $\mathrm{ext}(w') \in wp(T_{i-1})$.

Now we bound the change in potential $\phi_i - \phi_{i-1}$. Wings that did not change during the restructuring (i.e. those of type (i)) do not contribute to the potential difference. Also note, that i contributes to ϕ_{i-1}, but not to ϕ_i. Thus, we have for $1 \le i \le n$, assuming that $0 \log 0 = 0$, and denoting $f(x) = x \log(x)$:

$$\phi_i - \phi_{i-1} = \sum_{w' \in wp(A_i'')} \big(f(|w'| + |\mathrm{ext}(w')|) - f(|\mathrm{ext}(w')|) \big) - f(n_i - 1).$$

By simple manipulation, for $1 \le i \le n$:

$$\phi_i - \phi_{i-1} \ge \sum_{w' \in wp(A_i'')} f(|w'|) - f(n_i - 1).$$

By convexity of f, and observing that $|A_i''| = n_i - 2$, we have

$$\phi_i - \phi_{i-1} \ge |wp(A_i'')| \cdot f\left(\frac{n_i - 2}{|wp(A_i'')|}\right) - f(n_i - 1) = (n_i - 2) \cdot \log \frac{n_i - 2}{|wp(A_i'')|} - f(n_i - 1).$$

Lemma 23. *If R has right-depth m, and k leaves, then $|wp(R)| \le mk$.*

Proof: For a wing w, let $\ell(w)$ be any leaf in the subtree rooted at the node of maximum depth in the wing. Clearly, for any leaf ℓ there can be at most m wings w with $\ell(w) = \ell$. The claim follows. □

Thus, $|wp(A_i'')| \le n^{o(1)}$. Summing the potential differences over i, we get $\phi_n - \phi_0 = -n \log n \ge -\sum_{i=1}^{n} n_i \log (n^{o(1)}) - O(n)$. Denoting the total cost of algorithm $\mathcal{A}$ on the sequential access sequence as C, we obtain $C = \sum_{i=1}^{n} n_i = n \cdot \omega(1)$.

This shows that $\mathcal{A}$ does not satisfy the sequential access theorem.

6 Small Monotonicity-Depth and Local Algorithms

In this section we define a class of minimally self-adjusting BST algorithms that we call *local*. We show that an algorithm is local exactly if all after-trees it creates can be decomposed into constantly many monotone sets. Our definition of local algorithm is inspired by similar definitions by Subramanian [12] and Georgakopoulos and McClurkin [7]. Our locality criterion subsumes both previous definitions, apart from a technical condition not needed in these works: we require the transformation to bring the accessed element to the root. We require this (rather natural) condition in order to simplify the proofs. We mention that it can be removed at considerable expense in technicalities. Apart from this point, our definition of locality is more general: while existing local algorithms are oblivious to the global structure of the after-tree, our definition of local algorithm allows external global advice, as well as non-determinism.

Consider the before-path P and the after-tree A. A *decomposition* of the transformation $P \to A$ is a sequence of BSTs $(P = Q_0 \xrightarrow{P_0} Q_1 \xrightarrow{P_1} \dots \xrightarrow{P_{k-1}} Q_k = A)$, such that for all i, the tree Q_{i+1} can be obtained from the tree Q_i, by rearranging a path P_i contained in Q_i into a tree T_i, and linking all the attached subtrees in the unique way given by the element ordering. Clearly, every transformation has such a decomposition, since a sequence of rotations fulfills the requirement. The decomposition is *local* with window-size w, if it satisfies the following conditions:

(i) (start) $s \in P_0$, where s is the accessed element in P,
(ii) (progress) $P_{i+1} \setminus P_i \neq \emptyset$, for all i,
(iii) (overlap) $P_{i+1} \cap P_i \neq \emptyset$, for all i,
(iv) (no-revisit) $(P_i - P_{i+1}) \cap P_j = \emptyset$, for all $j > i + 1$,
(v) (window-size) $|P_i| \le w$, for some constant $w > 0$.

We call a minimally self-adjusting algorithm $\mathcal{A}$ *local*, if all the before-path $\rightarrow$ after-tree transformations performed by $\mathcal{A}$ have a local decomposition with constant-size window. The following theorem shows that local algorithms are exactly those that respect condition (ii) of Theorem 1 (proof omitted).

Theorem 24. *Let $\mathcal{A}$ be a minimally self-adjusting algorithm. (i) If $\mathcal{A}$ is local with window size w, then all the after-trees created by $\mathcal{A}$ can be partitioned into $2w$ monotone sets. (ii) If all the after-trees created by $\mathcal{A}$ can be partitioned into w monotone sets, then $\mathcal{A}$ is local with window-size w.*

Due to the relationship between monotone sets and locality of algorithms, we have

Theorem 25. *If a minimally self-adjusting BST algorithm $\mathcal{A}$ satisfies the access lemma with the SOL potential, then $\mathcal{A}$ can be made local.*

Open Questions: Does the family of algorithms described by Theorem 16 satisfy other efficiency-properties not captured by the access lemma? Properties studied in the literature include sequential access [13], deque [13,10], dynamic finger [4], or the elusive dynamic optimality [11].

One may ask whether locality is a necessary feature of all efficient BST algorithms. We have shown that some natural heuristics (e.g. path-balance or depth reduction) do not share this property, and thus do not satisfy the access lemma with the (rather natural) sum-of-logs potential function. It remains an open question, whether such "truly nonlocal" heuristics are necessarily bad, or if a different potential function could show that they are good.

Acknowledgement. The authors thank Raimund Seidel for suggesting the study of depth-reducing heuristics and for useful insights about BSTs and splay trees.

References

1. Allen, B., Munro, J.I.: Self-organizing binary search trees. J. ACM 25(4), 526–535 (1978)
2. Balasubramanian, R., Venkatesh, R.: Path balance heuristic for self-adjusting binary search trees. In: Proceedings of FSTTCS, pp. 338–348 (1995)
3. Chalermsook, P., Goswami, M., Kozma, L., Mehlhorn, K., Saranurak, T.: Self-adjusting binary search trees: What makes them tick? CoRR, abs/1503.03105 (2015)
4. Cole, R.: On the dynamic finger conjecture for splay trees. part ii: The proof. SIAM Journal on Computing 30(1), 44–85 (2000)
5. Demaine, E.D., Harmon, D., Iacono, J., Kane, D.M., Patrascu, M.: The geometry of binary search trees. In: SODA 2009, pp. 496–505 (2009)
6. Fox, K.: Upper Bounds for Maximally Greedy Binary Search Trees. In: Dehne, F., Iacono, J., Sack, J.-R. (eds.) WADS 2011. LNCS, vol. 6844, pp. 411–422. Springer, Heidelberg (2011)
7. Georgakopoulos, G.F., McClurkin, D.J.: Generalized template splay: A basic theory and calculus. Comput. J. 47(1), 10–19 (2004)
8. Lucas, J.M.: Canonical forms for competitive binary search tree algorithms. Tech. Rep. DCS-TR-250, Rutgers University (1988)
9. Munro, J.I.: On the competitiveness of linear search. In: Paterson, M. (ed.) ESA 2000. LNCS, vol. 1879, pp. 338–345. Springer, Heidelberg (2000)

10. Pettie, S.: Splay trees, Davenport-Schinzel sequences, and the deque conjecture. In: SODA 2008, pp. 1457–1467 (2008)
11. Sleator, D.D., Tarjan, R.E.: Self-adjusting binary search trees. J. ACM 32(3), 652–686 (1985)
12. Subramanian, A.: An explanation of splaying. J. Algorithms 20(3), 512–525 (1996)
13. Tarjan, R.E.: Sequential access in splay trees takes linear time. Combinatorica 5(4), 367–378 (1985)