Skip-Splay: Toward Achieving the Unified Bound in the BST Model

Jonathan C. Derryberry and Daniel D. Sleator

Computer Science Department
Carnegie Mellon University

Abstract. We present skip-splay, the first binary search tree algorithm known to have a running time that nearly achieves the unified bound. Skip-splay trees require only $O(m \lg \lg n + UB(\sigma))$ time to execute a query sequence $\sigma = \sigma_1 \dots \sigma_m$. The skip-splay algorithm is simple and similar to the splay algorithm.

1 Introduction and Related Work

Although the worst-case access cost for comparison-based dictionaries is $\Omega(\lg n)$, many sequences of operations are highly nonrandom, allowing tighter, instance-specific running time bounds to be achieved by algorithms that adapt to the input sequence. Splay trees [1] are an example of such an adaptive algorithm that operates within the framework of the binary search tree (BST) model [2], which essentially requires that all elements be stored in symmetric order in a rooted binary tree that can only be updated via rotations, and requires queried nodes to be rotated to the root. (BST algorithms that do not rotate to the root can usually be coerced into this model with just a constant factor of overhead.)

The two most general bounds proven for splay trees are the working set bound [1] and the dynamic finger bound [3], [4]. The working set bound shows that splay trees can have better than $O(\lg n)$ cost per operation when recently accessed elements are much more likely to be accessed than random elements, while the dynamic finger bound shows that splay trees have better than $O(\lg n)$ performance when each access is likely to be near the previous access.

Iacono later introduced the unified bound, which generalized both of these two bounds [5]. Roughly, a data structure that satisfies the unified bound has good performance for sequences of operations in which most accesses are likely to be near a recently accessed element. More formally, suppose the access sequence is $\sigma = \sigma_1 \dots \sigma_m$ and each access σ_j is a query to the set $\{1, \dots, n\}$ (we also use σ_j to refer to the actual element that is queried, as context suggests). The unified bound can be defined as follows:

$$UB(\sigma) = \sum_{j=1}^{m} \min_{j'<j} \lg(w(\sigma_{j'}, j') + |\sigma_{j'} - \sigma_j|), \tag{1}$$

where $w(x, j)$ is, at time j, the number of distinct elements including x that have been queried since the previous query to x, or n if no such previous query exists. For a more formal definition, see the definitions that precede Lemma 1.

F. Dehne et al. (Eds.): WADS 2009, LNCS 5664, pp. 194–205, 2009.

To achieve a running time of $O(m + UB(\sigma))$, Iacono introduced a data structure called the unified structure. The unified structure did not require amortization to achieve this bound, and was later improved by Bădoiu *et al.* to allow insertion and deletion [6]. The unified structure was comparison-based but did not adhere to the BST model. Thus, in addition to leaving open questions regarding how powerful the BST model was, it was not clear, for example, how to achieve the unified bound while keeping track of aggregate information on subsets of elements as can be done with augmented BSTs.

These unresolved issues motivate the question of whether a BST algorithm exists that achieves the unified bound. Achieving this goal contrasts with the separate pursuit of a provably dynamically optimal BST algorithm in that it is possible for a data structure that achieves the unified bound to have the trivial competitive ratio of $\Theta(\lg n)$ to an optimal BST algorithm. Conversely, prior to this work, even if a dynamically optimal BST algorithm had been found, it would not have been clear whether it satisfied the unified bound to within any factor that was $o(\lg n)$ since dynamic optimality by itself says nothing about actual formulaic bounds, and prior to this work no competitive factor better than $O(\lg n)$ was known for the cost of the optimal BST algorithm in comparison to the unified bound. See [7], [8], and [9] for progress on dynamic optimality in the BST model.

The skip-splay algorithm presented in this paper has three important qualities. First, it conforms to the BST model and has a running time of $O(m \lg \lg n + UB(\sigma))$, just an additive term of $O(\lg \lg n)$ per query away from the unified bound. Thus, skip-splay trees nearly close the gap between what is known to be achievable in the BST model and what is achieved by the unified structure. Second, the skip-splay algorithm is very simple. The majority of the complexity of our result resides in the analysis of skip-splaying, not in the design of the algorithm itself. The unified structure, though it avoids the additional $O(\lg \lg n)$ cost per query, is significantly more complicated than skip-splay trees. Finally, skip-splaying is almost identical to splaying, which suggests that a similar analysis, in combination with new insight, might be used to prove that splay trees satisfy the unified bound, at least to within some nontrivial multiplicative factor or additive term.

2 The Skip-Splay Algorithm

We assume for simplicity that a skip-splay tree T stores all elements of $\{1, \ldots, n\}$ where $n = 2^{2^{k-1}} - 1$ for some positive integer k, and that T is initially perfectly balanced. We mark as a splay tree root every node whose height (starting at a height of 1 for the leaves) is 2^i for $i \in \{0, \ldots, k-1\}$.[1] Note that the set of all of these splay trees partitions the elements of T.

[1] If we allow the ratio between the initial heights of successive roots to vary, we can achieve a parameterized running time bound, but in this version of the paper we use a ratio of 2 for simplicity.

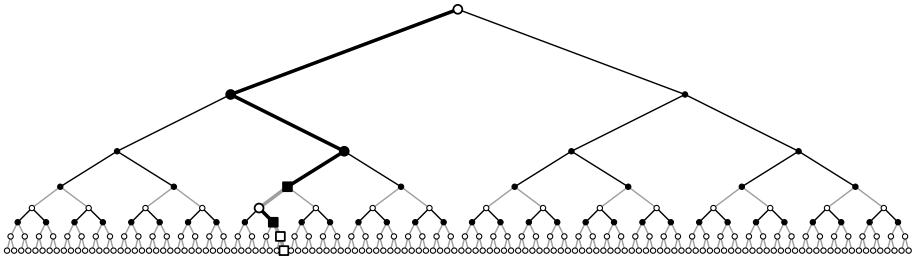

Fig. 1. An example of a four-level skip-splay tree T at the beginning of a query sequence. The white nodes are the roots of the splay trees that make up T, and the gray edges are never rotated. If the bottom element of the bold path is queried, then each of the boxed nodes is splayed to the root of its splay tree.

The following definitions will help us describe the algorithm more clearly:

1. Let T_i be the set of all keys x whose path to the root of T contains at most i root nodes, including x itself if x is marked as a root.
2. Define level i of T to be the set of keys x whose path to the root contains *exactly* i nodes. We will sometimes use the adjective "level-i" to refer to objects associated with level i in some way.
3. Let $tree(x)$ be the splay tree that contains x. Also, $tree(x)$ can represent the set of elements in $tree(x)$.

We assume that all operations are queries, and we use $\sigma = \sigma_1 \dots \sigma_m$ to denote the sequence of queries. To query an element σ_j, we first perform binary search through T to locate σ_j. Then, we splay σ_j to the root of $tree(\sigma_j)$ and transfer the relevant root marker to σ_j. If we are at the root of T, we terminate, else we "skip" to σ_j's new parent x and repeat this process by splaying x to the root of $tree(x)$. The cost of a query is defined to be the number of nodes on the access path to σ_j.[2] Figure 1 shows an example of what a skip splay tree looks like at the beginning of an access sequence and depicts how a query is performed.

Intuitively, skip-splaying is nearly competitive to the unified bound because if the currently queried element σ_j is near to a recently queried element σ_f, then many of the elements that are splayed while querying σ_j are likely to be the same as the ones that were splayed when σ_f was queried. Therefore, by the working set bound for splay trees, these splays should be fairly cheap. The analysis in Section 3 formalizes this intuition.

3 Proving Skip-Splay Runs in Time $O(m \lg \lg n + UB(\sigma))$

Our analysis in this section consists of three lemmas that together prove that skip-splay trees run in time $O(m \lg \lg n + UB(\sigma))$. The purpose of the first lemma

[2] Note that this algorithm can be coerced into the BST Model defined in [2] by rotating σ_j to the root and back down, incurring only a constant factor of additional cost.

is to decompose the cost of skip-splay trees into a series of "local working set costs" with one cost term for each level in T. The second lemma is the main step of the analysis and it uses the first lemma to prove that skip-splay trees satisfy a bound that is very similar to the unified bound, plus an additive $O(\lg \lg n)$ term. The third lemma shows that this similar bound is within a constant factor of the unified bound, so our main analytical result, that skip-splay trees run in $O(m \lg \lg n + UB(\sigma))$ time, follows immediately from these three lemmas.

In the first lemma and in the rest of this paper, we will use the following custom notation for describing various parts of T:

1. Let $\rho_k = 1$ and for $i < k$ let $\rho_i = 2^{2^{k-i-1}}$ so that $\rho_i = \rho_{i+1}^2$ for $i < k-1$. Note that if element $x \in T$ is in level i for $i < k$, then $|tree(x)| = \rho_i - 1$.
2. Let $R_i(x)$, the level-i region of $x \in T$ be defined as follows. First, define the offset $\delta_i = \delta \bmod \rho_i$, where δ is an integer that is arbitrary but fixed for all levels of T. (Our analysis will later make use of the fact that we can choose δ to be whatever we want.) Then, let $R_i(x) = R_i^*(x) \cap T$ where
$$R_i^*(x) = \left\{ \left\lfloor \tfrac{x+\delta_i}{\rho_i} \right\rfloor \rho_i - \delta_i, \ldots, \left\lfloor \tfrac{x+\delta_i}{\rho_i} \right\rfloor \rho_i - \delta_i + \rho_i - 1 \right\}.$$
Note that the level-i regions partition the elements of T and the level-$i+1$ regions are a refinement of the level-i regions. Two regions R and R' are said to be *adjacent* if they are distinct, occupy the same level, and their union covers a contiguous region of keyspace. Note that $|R_i(x)| = \rho_i$ if $R_i^*(x) \subseteq T$.
3. Let $\mathcal{R}_i(x)$, the level-i region set of x, be the set of level-i regions that are subsets of $R_{i-1}(x)$ with $\mathcal{R}_1(x)$ defined to be the set of all level-1 regions. Note that $|\mathcal{R}_i(x)| = \rho_i$ if $1 < i < k$ and $R_{i-1}^*(x) \subseteq T$.

Additionally, we give the following definitions of working set numbers and some auxiliary definitions that will also be helpful (these definitions assume we are working with a fixed query sequence σ):

1. Let $splays(j)$ be the set of elements that are splayed during query σ_j.
2. Let $p(x, j)$ represent the index of the previous access to x before time j. More formally, assuming such an access exists, let
$$p(x, j) = \max(\{1, \ldots, j-1\} \cap \{j' \mid \sigma_{j'} = x\}).$$
We define $p(x, j) = -n$ if the argument to max is the empty set.
3. Let $p'(x, j)$ represent the index of the previous access that resulted in a splay to x before time j. More formally, assuming such an access exists, let
$$p'(x, j) = \max(\{1, \ldots, j-1\} \cap \{j' \mid x \in splays(j')\}).$$
We define $p'(x, j) = -\rho_i$ if the argument to max is the empty set.
4. Let $p_i(x, j)$ represent the index of the previous access to region $R_i(x)$. More formally, assuming such an access exists, let
$$p_i(x, j) = \max(\{1, \ldots, j-1\} \cap \{j' \mid R_i(\sigma_{j'}) = R_i(x)\}).$$
We define $p_i(x, j) = -\rho_i$ if the argument to max is the empty set. Also, let $p_i(R, j)$ be equivalent to $p_i(x, j)$ if $R = R_i(x)$.

5. For $x \in T$, let $w(x,j)$ represent the number of elements queried since the previous access to x. More formally, if $p(x,j) > 0$ let

$$w(x,j) = \left|\left\{\sigma_{j'} \mid j' \in \{p(x,j), \ldots, j-1\}\right\}\right|.$$

   Else, if $p(x,j) \leq 0$ then let $w(x,j) = -p(x,j)$.
6. For $x \in T$, let $w'(x,j)$ represent the working set number of x within $tree(x)$ (i.e., the number of elements splayed in $tree(x)$ since the previous query resulting in a splay to x). More formally, if $p'(x,j) > 0$ let

$$w'(x,j) = \left|tree(x) \cap \bigcup_{j' \in \{p'(x,j), \ldots, j-1\}} splays(j')\right|.$$

   Else, if $p'(x,j) \leq 0$ then let $w'(x,j) = -p'(x,j)$.
7. For $x \in T$, let $w_i(x,j)$ represent the number of regions in $\mathcal{R}_i(x)$ that contain a query since the previous access to a member of $R_i(x)$. More formally, if $p_i(x,j) > 0$ let

$$w_i(x,j) = \left|\left\{R_i(\sigma_{j'}) \mid j' \in \{p_i(x,j), \ldots, j-1\}\right\} \cap \mathcal{R}_i(x)\right|.$$

   Else, if $p_i(x,j) \leq 0$ then let $w_i(x,j) = -p_i(x,j)$. Also, let $w_i(R,j)$ be equivalent to $w_i(x,j)$ if $R = R_i(x)$.
8. For $x \in T$, let $w'_i(x,j)$ be the working set number of x within $tree(x)$ that is reset whenever a query is executed to a region that could cause a splay of x. More formally, let $\mathcal{R}(x)$ be the set of up to three regions R such that a query to R can cause a splay of x. If $p_i(R,j) > 0$ for some $R \in \mathcal{R}(x)$ let

$$w'_i(x,j) = \left|tree(x) \cap \bigcup_{j' \in \{\max_{R \in \mathcal{R}(x)} p_i(R,j), \ldots, j-1\}} splays(j')\right|.$$

   Else, if $p_i(R,j) \leq 0$ for all $R \in \mathcal{R}(x)$ then let $w'_i(x,j) = \rho_i$. Note that $w'_i(x,j) \leq 3w_i(R,j) + 1$ for $R \in (\mathcal{R}(x) \cap \mathcal{R}_i(x))$ because accesses to a region in $\mathcal{R}_i(x)$ can result in splays of at most three different elements of $tree(x)$, and at most one, the minimum element of $tree(x)$, can be splayed as the result of a query to another level-i region set. Also, note that $w'_i(x,j) \leq w'(x,j)$.

In the proof of the first lemma, we will be making use of the working set theorem in Sleator and Tarjan's original splay tree paper [1], which shows that the cost of a query sequence σ on an individual splay tree, for sufficiently large n, is bounded by $c_s(n \lg n + \sum_{j=1}^{m} \lg(w(\sigma_j, j) + 1))$, for some constant c_s. For simplicity, we assume we are starting with a minimum potential arrangement of each splay tree, so this simplifies to $\sum_{j=1}^{m} c_s \lg(w(\sigma_j, j)+1)$. In order to make the analysis in Lemma 2 simpler, we move beyond simply associating this working set cost with each splay that is executed in T by proving the following lemma.

Lemma 1. *For query sequence σ in a skip-splay tree T with k levels, the amortized cost of query σ_j is*

$$O\left(k + \sum_{i=1}^{k} \lg w_i(\sigma_j, j)\right). \tag{2}$$

Proof. By the definition of the skip-splay algorithm and the working set theorem for splay trees, the amortized cost of query σ_j is $\sum_{x \in splays(j)} w'(x, j)$, suppressing multiplicative and additive constants. To prove Lemma 1, we will do further accounting for the cost of a query σ_j and focus on the cost associated with an arbitrary level i of T.

Note that at level i during query σ_j, one of three cases occurs with regard to which level-i node, if any, is splayed. First, if σ_j resides in a strictly shallower level than i, then no splay is performed in level i. Second, if σ_j resides within level i, then σ_j is splayed in level i. Third, if σ_j resides in a deeper level than i, then either the predecessor or the successor of σ_j in level i is splayed. (We know that at least one of these two nodes exists and is on the access path in this case.) We will use the following potential function on T to prove that the bound in Equation 2 holds regardless of which of these three cases occurs:

$$\Phi(T, j) = \phi_1(T, j) + \phi_2(T, j), \tag{3}$$

where

$$\phi_1(T, j) = \sum_{x \in T} (\lg w'(x, j+1) - \lg w'_i(x, j+1)) \tag{4}$$

and

$$\phi_2(T, j) = \sum_{(x,y) \in A} |\lg w'_i(x, j+1) - \lg w'_i(y, j+1)|, \tag{5}$$

where A is the set of pairs of level-i nodes (x, y) such that x is the maximum element in $tree(x)$, y is the minimum element in $tree(y)$, and there are no other level-i elements between x and y. For succinctness below, define $\Delta\Phi(T, j)$ to be $\Phi(T, j) - \Phi(T, j-1)$ and define $\Delta\phi_1(T, j)$ and $\Delta\phi_2(T, j)$ analogously.

First, notice that the cost of the splay, if any, that is performed on node x at level i is offset by the change in potential of $\lg w'(x, j+1) - \lg w'(x, j) = -\lg w'(x, j)$. Note that this ignores the difference $\lg w'_i(x, j) - \lg w'_i(x, j+1) = \lg w'_i(x, j)$, which will be accounted for below.

Second, define $\Delta_+\Phi(T, j)$ to be the sum of the positive terms of $\Delta\Phi(T, j)$ plus $\lg w'_i(x, j)$ in the case in which some node x is splayed during query σ_j. We will show that regardless of whether a splay is performed in level i during query σ_j, it is true that $\Delta_+\Phi(T, j)$ is at most $4\lg(3w_i(\sigma_j, j) + 1) + 2$.

To see this, let Y be the set of up to three level-i nodes that can be splayed while accessing members of the region $R_i(\sigma_j)$, and notice that if a node x is splayed at level i during query σ_j then $x \in Y$. Note that the only positive terms of $\Delta_+\Phi(T, j)$ from $\Delta\phi_1(T, j)$ are the ones that use some member of Y as an argument. This is true because $\lg w'(z, j+1) - \lg w'(z, j) \leq \lg w'_i(z, j+1) - \lg w'_i(z, j)$

for $z \in T \backslash Y$ since $w'(z,j) \geq w'_i(z,j)$ and $w'(z,j+1) - w'(z,j) \leq w'_i(z,j+1) - w'_i(z,j)$. Further, note that $\Delta_+\Phi(T,j)$ contains at most two terms from $\Delta\phi_2(T,j)$ that do not use some member of Y as an argument, and these two terms are at most 1 each.

Now, we consider the following two cases. All additional cases are either similar to or simpler than these two cases. First, suppose that Y contains two elements $y_1 < y_2$ and $tree(y_1) \neq tree(y_2)$. Note that in this case we know that $\mathcal{R}_i(\sigma_j) = \mathcal{R}_i(y_1)$. Then,

$$\begin{aligned}\Delta_+\Phi(T,j) &\leq \lg w'_i(y_1,j) + \lg w'_i(y_2,j) - |\lg w'_i(y_1,j) - \lg w'_i(y_2,j)| + 2 \\ &\leq 2\lg w'_i(y_1,j) + 2 \\ &\leq 2\lg(3w_i(\sigma_j,j)+1) + 2.\end{aligned}$$

Second, suppose that Y contains three elements $y_1 < y_2 < y_3$ that all reside in the same splay tree T', suppose y_3 is the maximum element of T', and let z be the successor of y_3 among the level-i elements (assuming z exists in this case). Using the fact that $|\lg w'_i(y_3,j+1) - \lg w'_i(z,j+1)| = \lg w'_i(z,j+1) = \lg w'_i(z,j)$ and the fact that $\mathcal{R}_i(y_1) = \mathcal{R}_i(y_2) = \mathcal{R}_i(y_3) = \mathcal{R}_i(\sigma_j)$, we have

$$\begin{aligned}\Delta_+\Phi(T,j) &\leq \sum_{q=1}^{3} \lg w'_i(y_q,j) + \lg w'_i(z,j) - |\lg w'_i(y_3,j) - \lg w'_i(z,j)| + 2 \\ &\leq \lg w'_i(y_1,j) + \lg w'_i(y_2,j) + 2\lg w'_i(y_3,j) + 2 \\ &\leq 4\lg(3w_i(\sigma_j,j)+1) + 2.\end{aligned}$$

□

We note that the potential function used in Lemma 1 starts at its minimum value and the splay trees also start at their minimum potential configuration. Therefore, the sum of the amortized costs of each query, according to Lemma 1, is an upper bound on the cost of the sequence. Using Lemma 1, we can prove a bound that is similar to the unified bound, plus an additive $O(\lg\lg n)$ term per query. This bound differs from the unified bound in that the working set portion of the cost consists not of the number of *elements* accessed since the previous query to the relevant element, but of the number of *queries* since the previous query to the relevant element. Before we prove this bound, we give the following definitions, which will be useful in formally describing the bound and proving it:

1. Let f_j represent the element $\sigma_{j'}$ such that
$$j' = \operatorname*{argmin}_{j''<j} \lg(w(\sigma_{j''},j) + |\sigma_j - \sigma_{j''}|).$$
Intuitively, f_j represents the "finger" for query σ_j because it represents the previously-queried element that yields the smallest unified bound value for query σ_j.
2. For $x \in T$, let $t(x,j)$ represent the number of *queries* (rather than distinct elements accessed) since the previous access to x. More formally, let
$$t(x,j) = |\{p(x,j),\ldots,j-1\}| = j - p(x,j).$$
Note that the above definition handles the case in which $p(x,j) \leq 0$.

3. For $x \in T$, let $t_i(x, j)$ represent the number of queries to all members of $\mathcal{R}_i(x)$ since the previous access to a member of $R_i(x)$. More formally, let

$$t_i(x, j) = \left| \left\{ j' \in \{\max(1, p_i(x, j)), \ldots, j-1\} \mid R_i(\sigma_{j'}) \in \mathcal{R}_i(x) \right\} \right|,$$

with an additional $-p_i(x, j)$ added if $p_i(x, j) \leq 0$.

4. For $x \in T$, let $\hat{t}_i(x, j)$ represent the number of queries to all members of $\mathcal{R}_i(x)$ since the previous access to x. More formally, let

$$\hat{t}_i(x, j) = \left| \left\{ j' \in \{\max(1, p(x, j)), \ldots, j-1\} \mid R_i(\sigma_{j'}) \in \mathcal{R}_i(x) \right\} \right|,$$

with an additional ρ_i^2 added if $p(x, j) \leq 0$. Note that $\hat{t}_1(x, j) \leq t(x, j) + 1$ by definition.

Next, we define $UB'(\sigma)$, a variant of the unified bound, as

$$UB'(\sigma) = \sum_{j=1}^{m} \lg(t(f_j, j) + |\sigma_j - f_j|), \tag{6}$$

and we are ready to proceed with our second lemma.

Lemma 2. *Executing the skip-splay algorithm on query sequence $\sigma = \sigma_1 \ldots \sigma_m$ costs time $O(m \lg \lg n + UB'(\sigma))$.*

Proof. In this proof, we will be making use of the bound in Lemma 1 with a randomly chosen offset δ that is selected uniformly at random from $\{0, \ldots, \rho_1 - 1\}$. We will use induction on the number of levels i from the top of the tree while analyzing the expected amortized cost of an arbitrary query σ_j. In the inductive step, we will prove a bound that is similar to the one in Lemma 2, and this similar bound will cover the cost associated with levels i and deeper. Even though we are directly proving the inductive step in expectation only, because the bound in Lemma 1 is proven for all values of δ, we know that there exists at least one value of δ such that the bound holds *without* using randomization if we amortize over the entire query sequence. Therefore, the *worst-case* bound on the *total* cost of the access sequence in Lemma 2 will follow.

Our inductive hypothesis is that the cost of skip-splaying σ_j that is associated with levels $i + 1$ and deeper according to Lemma 1 is at most

$$\alpha \lg \hat{t}_{i+1}(f_j, j) + \beta \lg \min(1 + |\sigma_j - f_j|^2, \rho_{i+1}) + \gamma(k - i), \tag{7}$$

where k, as before, represents the number of levels of splay trees in T.

We choose levels k and $k - 1$ to be our base cases. The inductive hypothesis is trivially true for these base cases as long as we choose the constants appropriately. Also, the bound for the inductive hypothesis at level 1, summed over all queries, is $O(m \lg \lg n + UB'(\sigma))$, so proving the inductive step suffices to prove the lemma.

To prove the inductive step, we assume Equation 7 holds for level $i+1$ and use this assumption to prove the bound for level i. Thus, our goal is to prove the following bound on the cost that Lemma 1 associates with query σ_j for levels i and deeper:

$$\alpha \lg \hat{t}_i(f_j, j) + \beta \lg \min(1 + |\sigma_j - f_j|^2, \rho_i) + \gamma(k - i + 1). \tag{8}$$

As a starting point for the proof of the inductive step, Lemma 1 in addition to the inductive hypothesis allows us to prove an upper bound of

$$\lg w_i(\sigma_j, j) + \alpha \lg \hat{t}_{i+1}(f_j, j) + \beta \lg \min(1 + |\sigma_j - f_j|^2, \rho_{i+1}) + \gamma(k - i), \tag{9}$$

where we have suppressed the constant from Lemma 1 multiplying $\lg w_i(\sigma_j, j)$.

Our proof of the inductive step consists of three cases. First, if $|\sigma_j - f_j|^2 \geq \rho_i$, then substituting ρ_i for ρ_{i+1} increases the bound in Equation 9 by

$$\lg \rho_i - \lg \rho_{i+1} = \lg \left(\tfrac{\rho_i}{\rho_{i+1}} \right) = \lg (\rho_{i+1}) = \lg \left(\rho_i^{1/2} \right) \geq \lg \left(w_i(\sigma_j, j)^{1/2} \right), \tag{10}$$

which offsets the elimination of the cost $\lg w_i(\sigma_j, j)$ as long as $\beta \geq 2$. The other substitutions only increase the bound, so for this case we have proved the inductive step.

Second, if $|\sigma_j - f_j|^2 < \rho_i$ and $R_i(\sigma_j) \neq R_i(f_j)$, then we simply pay $\lg w_i(\sigma, j)$ which is at most $\lg \rho_i$. However, we note that the probability of this occurring for a random choice of δ is at most $\rho_i^{1/2}/\rho_i = \rho_i^{-1/2}$, so the expected cost resulting from this case is at most $\rho_i^{-1/2} \lg \rho_i$, which is at most a constant, so it can be covered by γ.

The third and most difficult case occurs when $|\sigma_j - f_j|^2 < \rho_i$ and $R_i(\sigma_j) = R_i(f_j)$, and we will spend the rest of the proof demonstrating how to prove the inductive step for this case. First, we note that $\lg t_i(f_j, j) \geq \lg w_i(f_j, j) = \lg w_i(\sigma_j, j)$, so we can replace $\lg w_i(\sigma_j, j)$ with $\lg t_i(f_j, j)$ and ρ_{i+1} with ρ_i in Equation 9 without decreasing the bound and prove a bound of

$$\lg t_i(f_j, j) + \alpha \lg \hat{t}_{i+1}(f_j, j) + \beta \lg \min(1 + |\sigma_j - f_j|^2, \rho_i) + \gamma(k - i). \tag{11}$$

It remains only to eliminate the term $\lg t_i(f_j, j)$ by substituting $\hat{t}_i(f_j, j)$ for $\hat{t}_{i+1}(f_j, j)$ while incurring an additional amortized cost of at most a constant so that it can be covered by γ.

Observe that if σ_j satisfies

$$\hat{t}_{i+1}(f_j, j) \leq \tfrac{\hat{t}_i(f_j, j)}{t_i(f_j, j)^{\frac{1}{2}}}, \tag{12}$$

then we have an upper bound of

$$\lg t_i(f_j, j) + \alpha(\lg \hat{t}_i(f_j, j) - \tfrac{\lg t_i(f_j, j)}{2}) + \beta \lg \min(1 + |\sigma_j - f_j|^2, \rho_i) + \gamma(k - i), \tag{13}$$

which would prove the inductive step if $\alpha \geq 2$. However, it is possible that $\hat{t}_{i+1}(f_j, j)$ does not satisfy the bound in Equation 12. In this latter case, we

pessimistically assume that we must simply pay the additional $\lg t_i(f_j, j)$. In the rest of the proof, we show that the amortized cost of such cases is at most a constant per query in this level of the induction, so that it can be covered by the constant γ.

We first give a few definitions that will make our argument easier. A query σ_b is *R-local* if $R_i(\sigma_b) = R$. Further, if σ_b is R-local and satisfies $R_i(f_b) = R$ as well as the bound $\hat{t}_{i+1}(f_b, b) > \hat{t}_i(f_b, b)/t_i(f_b, b)^{\frac{1}{2}}$, then we define σ_b also to be *R-dense*. Note that if σ_b is R-dense then $p(f_b, b) > 0$. Finally, if σ_b additionally satisfies the inequality $\tau < t_i(f_b, b) \leq 2\tau$, then we define σ_b also to be *R-τ-bad*. Notice that all queries that have an excess cost at level i due to being in this third case and not meeting the bound in Equation 12 are R-τ-bad for some level-i region R and some value of τ (actually a range of values τ).

Our plan is to show that the ratio of R-τ-bad queries to R-local queries is low enough that the sum of the excess costs associated with the R-τ-bad queries can be spread over the R-local queries so that each R-local query is only responsible for a constant amount of these excess costs. Further, we show that if we partition the R-dense queries by successively doubling values of τ, with some constant lower cutoff, then each R-local query's share of the cost is exponentially decreasing in $\lg \tau$, so each R-local query bears only a constant amortized cost for the excess costs of all of the R-dense queries. Lastly, note that in our analysis below we are only amortizing over R-local queries for some specific but arbitrary level-i region R, so we can apply the amortization to each level-i region separately without interference.

To begin, we bound the cost associated with the R-τ-bad queries for arbitrary level-i region R and constant τ as follows. Let σ_b be the latest R-τ-bad query. First, note that the number of R-τ-bad queries σ_a where $a \in \{p(f_b, b)+1, \ldots, b\}$ is at most $\hat{t}_i(f_b, b)/\tau$ because there are $\hat{t}_i(f_b, b)$ queries to $\mathcal{R}_i(f_b)$ in that time period, and immediately prior to each such σ_a, the previous $\tau - 1$ queries to $\mathcal{R}_i(f_b)$ are all outside of R so that $t_i(f_a, a) \geq \tau$. Second, note that because σ_b was chosen to be R-τ-bad we have

$$\hat{t}_{i+1}(f_b, b) > \frac{\hat{t}_i(f_b,b)}{t_i(f_b,b)^{1/2}} \geq \frac{\hat{t}_i(f_b,b)}{(2\tau)^{1/2}}. \tag{14}$$

Thus, the ratio of the number of R-local queries in this time period, $\hat{t}_{i+1}(f_b, b)$, to the number of R-τ-bad queries in this time period is strictly greater than

$$\frac{\hat{t}_i(f_b,b)}{(2\tau)^{1/2}} \cdot \frac{\tau}{\hat{t}_i(f_b,b)} = (\tfrac{\tau}{2})^{1/2}. \tag{15}$$

The constraint that $t_i(f_a, a) \leq 2\tau$ for each of the aforementioned R-τ-bad queries σ_a implies that the excess level-i cost of each is at most $\lg(2\tau)$, so we charge each R-local query with a time index in $\{p(f_b, b)+1, \ldots, b\}$ a cost of $\lg(2\tau)/(\frac{\tau}{2})^{1/2}$ to account for the R-τ-bad queries that occur during this time interval. Notice that we can iteratively apply this reasoning to cover the R-τ-bad queries with time indices that are at most $p(f_b, b)$ without double-charging any R-local query.

To complete the argument, we must account for all R-dense queries, not just the R-τ-bad ones for some particular value of τ. To do this, for all R-dense queries

σ_j such that $t_i(f_j, j) \leq \tau_0$, for some constant τ_0, we simply charge a cost of $\lg \tau_0$ to γ. Next, let $\tau_q = 2^q \tau_0$ for integer values $q \geq 0$. From above, we have an upper bound on the amortized cost of the R-τ_q-bad queries of $\lg(2^{q+1}\tau_0)/(2^{q-1}\tau_0)^{1/2}$, so the sum over all values of q is at most a constant and can be covered by γ. □

To complete the argument that skip-splay trees run in $O(m \lg \lg n + UB(\sigma))$ time, it suffices to show that $UB'(\sigma)$ is at most a constant factor plus a linear term in m greater than $UB(\sigma)$. Thus, the following lemma completes the proof that skip-splay trees run in time $O(m \lg \lg n + UB(\sigma))$.

Lemma 3. *For query sequence $\sigma = \sigma_1 \ldots \sigma_m$, the following inequality is true:*

$$\sum_{j=1}^{m} \lg(t(f_j, j) + |\sigma_j - f_j|) \leq \tfrac{m\pi^2 \lg e}{6} + \lg e + \sum_{j=1}^{m} 2 \lg(w(f_j, j) + |\sigma_j - f_j|). \quad (16)$$

Proof. To begin, we give a new definition of a working set number that is a hybrid between $w(f_j, j)$ and $t(f_j, j)$ for arbitrary time index j. Let $h_i(f_j, j) = \max(w(f_j, j)^2, \min(t(f_j, j), j - i))$. Note that $\lg h_m(f_j, j) = 2 \lg w(f_j, j)$ and $h_{-n}(f_j, j) \geq t(f_j, j)$ for all j. Also, note that if $p(f_j, j) > 0$ then $\lg h_{-n}(f_j, j) - \lg h_0(f_j, j) = 0$, else if $p(f_j, j) \leq 0$, which is true for at most n queries, then $\lg h_{-n}(f_j, j) - \lg h_0(f_j, j) \leq \lg(n^2 + n) - \lg(n^2) \leq \frac{\lg e}{n}$.

Next, note that $\lg h_i(f_j, j) - \lg h_{i+1}(f_j, j) = 0$ if $i \geq j$ or $t(f_j, j) \leq j - i - 1$ and for all j we have $\lg h_i(f_j, j) - \lg h_{i+1}(f_j, j) \leq \frac{\lg e}{w(f_j, j)^2}$. Also, we know that the number of queries for which $i < j$, $t(f_j, j) \geq j - i$, and $w(f_j, j) \leq w_0$ is at most w_0 for $w_0 \in \{1, \ldots, n\}$. This is true because each such query is to a distinct element since they all use a finger that was last queried at a time index of at most i (if two of these queries were to the same element, then the second query could use the first as a finger). If there were $w_0 + 1$ such queries, the latest such query σ_ℓ would have $w(f_\ell, j) \geq w_0 + 1$ because of the previous w_0 queries after time i to distinct elements, a contradiction. Therefore,

$$\sum_{j=1}^{m} (\lg h_i(f_j, j) - \lg h_{i+1}(f_j, j)) \leq \sum_{k=1}^{n} \tfrac{\lg e}{k^2} \leq \tfrac{\pi^2 \lg e}{6},$$

so that

$$\sum_{j=1}^{m} (\lg t(f_j, j) - 2 \lg w(f_j, j)) \leq \sum_{j=1}^{m} (\lg h_{-n}(f_j, j) - \lg h_m(f_j, j)) \leq \tfrac{m\pi^2 \lg e}{6} + \lg e.$$

The fact that $\lg(t(f_j, j) + d) - 2 \lg(w(f_j, j) + d) \leq \lg t(f_j, j) - 2 \lg w(f_j, j)$ for all j and non-negative d completes the proof. □

4 Conclusions and Future Work

The ideal improvement to this result is to show that splay trees satisfy the unified bound with a running time of $O(m + UB(\sigma))$. However, achieving this ideal result

could be extremely difficult since the only known proof of the dynamic finger theorem is very complicated, and the unified bound is stronger than the dynamic finger bound.

In light of this potential difficulty, one natural path for improving this result is to apply this analysis to splay trees, perhaps achieving the same competitiveness to the unified bound as skip-splay trees. Intuitively, this may work because the skip-splay algorithm is essentially identical to splaying except a few rotations are skipped to keep the elements of the tree partitioned into blocks with a particular structure that facilitates our analysis.

Additionally, it may be possible to design a different BST algorithm and show that it meets the unified bound, which would prove that we do not need to leave the BST model, and the perks such as augmentation that it provides, to achieve the unified bound. If such an algorithm is to be similar to skip-splaying, it must mix the splay trees together so that all nodes can reach constant depth.

To summarize the clearest paths for related future work, it would be significant progress to show that splay trees meet the unified bound to within any factor that is $o(\lg n)$, or to show that some BST algorithm achieves the unified bound to within better than an additive $O(\lg \lg n)$ term.

References

1. Sleator, D.D., Tarjan, R.E.: Self-adjusting binary search trees. Journal of the ACM 32, 652–686 (1985)
2. Wilber, R.: Lower bounds for accessing binary search trees with rotations. SIAM Journal on Computing 18(1), 56–67 (1989)
3. Cole, R., Mishra, B., Schmidt, J.P., Siegel, A.: On the dynamic finger conjecture for splay trees, part I: Splay sorting log n-block sequences. SIAM Journal on Computing 30(1), 1–43 (2000)
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. Iacono, J.: Alternatives to splay trees with o(log n) worst-case access times. In: Proceedings of the 12th ACM-SIAM Symposium on Discrete Algorithms, Philadelphia, PA, USA, pp. 516–522. Society for Industrial and Applied Mathematics (2001)
6. Bădoiu, M., Cole, R., Demaine, E.D., Iacono, J.: A unified access bound on comparison-based dynamic dictionaries. Theoretical Computer Science 382(2), 86–96 (2007)
7. Demaine, E.D., Harmon, D., Iacono, J., Pătraşcu, M.: Dynamic optimality—almost. SIAM Journal on Computing 37(1), 240–251 (2007)
8. Wang, C.C., Derryberry, J., Sleator, D.D.: O(log log n)-competitive dynamic binary search trees. In: Proceedings of the 17th ACM-SIAM Symposium on Discrete Algorithms, pp. 374–383. ACM, New York (2006)
9. Georgakopoulos, G.F.: Chain-splay trees, or, how to achieve and prove loglogn-competitiveness by splaying. Information Processing Letters 106(1), 37–43 (2008)