Upper Bounds for Maximally Greedy Binary Search Trees

Kyle Fox

Department of Computer Science, University of Illinois, Urbana-Champaign
kylefox2@illinois.edu

Abstract. At SODA 2009, Demaine et al. presented a novel connection between binary search trees (BSTs) and subsets of points on the plane. This connection was independently discovered by Derryberry et al. As part of their results, Demaine et al. considered GreedyFuture, an offline BST algorithm that greedily rearranges the search path to minimize the cost of future searches. They showed that GreedyFuture is actually an online algorithm in their geometric view, and that there is a way to turn GreedyFuture into an online BST algorithm with only a constant factor increase in total search cost. Demaine et al. conjectured this algorithm was dynamically optimal, but no upper bounds were given in their paper. We prove the first non-trivial upper bounds for the cost of search operations using GreedyFuture including giving an access lemma similar to that found in Sleator and Tarjan's classic paper on splay trees.

1 Introduction

The ***dynamic optimality conjecture*** states that given a sequence of successful searches on an n-node binary search tree, the number of nodes accessed by splay trees is at most a constant times the number of node accesses and rotations performed by the optimal algorithm *for that sequence*. Sleator and Tarjan gave this conjecture in their paper on splay trees in which they showed $O(\log n)$ amortized performance as well as several other upper bounds [12]. Proving the dynamic optimality conjecture seems very difficult. There is no known polynomial time algorithm for finding an optimal BST in the offline setting where we know all searches in advance,[1] and this conjecture states that splaying is a simple solution to the online problem.

Until recently, there has been little progress made directly related to this conjecture. Wilber gave two lower bounds on the number of accesses needed for any given search sequence [16]. There are a handful of online BST algorithms that are $O(\log \log n)$-competitive [5,15,14,1], but no upper bound is known for the competitiveness of splay trees except the trivial $O(\log n)$.

[1] In fact, the exact optimization problem becomes NP-hard if we must access an arbitrary number of specified nodes during each search [4].

F. Dehne, J. Iacono, and J.-R. Sack (Eds.): WADS 2011, LNCS 6844, pp. 411–422, 2011.

1.1 A Geometric View

Recently, Demaine et al. introduced a new way of conceptualizing BSTs using geometry [4]. A variant of this model was independently discovered by Derryberry et al. [6]. In the geometric view, BST node accesses are represented as points (x, y) where x denotes the rank of the accessed node and y represents which search accessed the node. A pair of points a and b in point set P are called *arborally satisfied* if they lie on the same horizontal or vertical line, or if the closed rectangle with corners a and b contains another point from P. The family of arborally satisfied point sets corresponds exactly to BST accesses when rotations upon accessed nodes are allowed [4].

By starting with a point set X that represents the points a BST must access to complete searches in a given search sequence S, we can describe an optimal BST algorithm for S as a minimum superset of X that is arborally satisfied [4]. This correspondence between BSTs and arborally satisfied supersets allows us to focus on algorithms strictly in the geometric view. Additionally, it is possible to show lower bounds for the BST model by showing the same for the geometric model. Demaine et al. take advantage of this fact to show a class of lower bounds that supersede the lower bounds of Wilber [16,4]. Further, it is possible to describe an *online* version of the arborally satisfied superset problem and transform solutions to this problem into online BST algorithms with at most a constant factor increase in cost [4].

1.2 Being Greedy

Demaine et al. also consider an offline BST algorithm they call GreedyFuture, originally proposed by Lucas [10] and Munro [11]. GreedyFuture only touches nodes on the search path, and then rearranges the search path in order to greedily minimize the time for upcoming searches.

The worst-case example known for the competitiveness of GreedyFuture is a complete binary search tree with searches performed in bit-reversal order upon the leaves [11]. GreedyFuture has an amortized cost of $\lg n$ per search on this sequence. The optimal algorithm rotates the leaves closer to the root and obtains an amortized cost of $\lg \frac{n}{2} + o(1)$. Given a search sequence of length m, let OPT be the total cost of the optimal algorithm for that sequence. Demaine et al. conjecture that GreedyFuture is $O(1)$-competitive. In fact, the bit-reversal example suggests that the cost of GreedyFuture is at most $\mathrm{OPT} + m$; it appears optimal within an *additive term*.

Surprisingly, Demaine et al. showed that GreedyFuture, an offline algorithm that uses very strong knowledge about the future, is actually an online algorithm in the geometric model [4]. Recall that online algorithms in the geometric model correspond to online algorithms in the BST model with essentially the same cost. If GreedyFuture is actually an offline dynamically optimal BST algorithm as it appears to be, then there exists an *online* dynamically optimal BST algorithm.

1.3 Our Contributions

Despite the apparent optimality of the GreedyFuture algorithm, nothing was known about its amortized behavior when Demaine et al. wrote their report. We provide the first theoretical evidence that GreedyFuture is an optimal algorithm in the following forms:

- An access lemma similar to that used by Sleator and Tarjan for splay trees [12]. This lemma implies several upper bounds including $O(\log n)$ amortized performance.
- A sequential access theorem that states GreedyFuture takes linear time to access all nodes in order starting from any arbitrary BST.

We heavily use the geometric model of Demaine et al. to prove the access lemma while focusing directly on BSTs to prove the sequential access theorem. It is our hope that these results will create further interest in studying GreedyFuture as its structural properties seem well suited for further theoretical analysis (the proof of the sequential access theorem takes only a page). Additionally, the proof of the access lemma may provide additional insight into other algorithms running in the geometric model.

1.4 A Note on Independent Work

John Iacono and Mihai Pătraşcu have discovered a similar access lemma to that given here using different proof techniques from those shown below. The author learned about their work via personal correspondence with them and Erik Demaine well into performing the research contained in this report. Their results have never been published.

Additionally, the author became aware of work by Goyal and Gupta [8] after initially writing this report. They show GreedyFuture has $O(\log n)$ amortized performance. This result appears in our paper as Corollary 2. As in our proof, they use the geometric model, but they do not use a potential function as we do to prove a more general access lemma.

2 Arboral and Geometric Models of BSTs

2.1 The Arboral Model

We will consider the same BST model used by Demaine et al. [4]. We consider only successful searches and not insertions or deletions. Let n and m be the number of elements in the search tree and the number of searches respectively. We assume the elements have distinct keys in $\{1, \ldots, n\}$.

Given a BST T_1, a subtree τ of T_1 containing the root, and a tree τ' on the same nodes as τ, we say T_1 can be ***reconfigured*** by an operation $\tau \rightarrow \tau'$ to another BST T_2 if T_2 is identical to T_1 except for τ being replaced by τ'. The cost of the reconfiguration is $|\tau| = |\tau'|$.

Given a search sequence $S = \langle s_1, s_2, \ldots, s_m \rangle$, we say a BST algorithm ***executes*** S by an execution $E = \langle T_0, \tau_1 \rightarrow \tau'_1, \ldots, \tau_m \rightarrow \tau'_m \rangle$ if all reconfigurations are performed on subtrees containing the root, and $s_i \in \tau_i$ for all i.

For $i = 1, 2, \ldots, m$, define T_i to be T_{i-1} with the reconfiguration $\tau_i \to \tau_i'$. The cost of execution E is $\sum_{i=1}^{m} |\tau_i|$.

As explained by Demaine et al. [4], this model is constant-factor equivalent to other reasonable BST models such as those by Wilber and Lucas [16,10].

2.2 The Geometric Model

We now turn our focus to the geometric model as given by Demaine et al. [4]. Define a ***point*** p to be a point in 2D with integer coordinates $(p.x, p.y)$ such that $1 \leq p.x \leq n$ and $1 \leq p.y \leq m$. Let $\square ab$ denote the closed axis-aligned rectangle with corners a and b.

A pair of points (a, b) (or their induced rectangle $\square ab$) is ***arborally satisfied*** with respect to a point set P if (1) a and b are orthogonally collinear (horizontally or vertically aligned), or (2) there is at least one point from $P \setminus \{a, b\}$ in $\square ab$. A point set P is arborally satisfied if all pairs of points in P are arborally satisfied with respect to P. See Fig. 1 and Fig. 2.

As explained in [4], there is a one-to-one correspondence between BST executions and arborally satisfied sets of points. Let the ***geometric view*** of a BST execution E be the point set $P(E) = \{(x, y) | x \in \tau_y\}$. The point set $P(E)$ for any BST execution E is arborally satisfied [4]. Further, for any arborally satisfied point set X, there exists a BST execution E with $P(E) = X$ [4].

Let the ***geometric view*** of an access sequence S be the set of points $P(S) = \{(s_1, 1), (s_2, 2), \ldots, (s_m, m)\}$. The above facts suggest that finding an optimal BST algorithm for S is equivalent to finding a minimum cardinality arborally satisfied superset of S. Due to this equivalence with BSTs, we will refer to values in $\{1, \ldots, n\}$ as ***elements***.

Naturally, we may want to use the geometric model to find dynamically optimal *online* BST algorithms. The ***online arborally satisfied superset*** (online ASS) problem is to design an algorithm that receives a sequence of points $\langle (s_1, 1), (s_2, 2), \ldots, (s_m, m) \rangle$ incrementally. After receiving the ith point (s_i, i), the algorithm must output a set P_i of points on the line $y = i$ such that $\{(s_1, 1), (s_2, 2), \ldots, (s_i, i)\} \cup P_1 \cup P_2 \cup \cdots \cup P_i$ is arborally satisfied. The cost of the algorithm is $m + \sum_{i=1}^{m} |P_i|$.

We say an online ASS algorithm performs a ***search*** at time i when it outputs the set P_i. Further, we say an online ASS algorithm ***accesses*** x at time i if (x, i) is included in the input set of points or in P_i. The (non-amortized) cost of a search at time i is $|P_i| + 1$.

Unfortunately, the algorithm used to create a BST execution from an arborally satisfied point set requires knowledge about points above the line $y = i$ to construct T_i [4]. We are not able to go directly from a solution to the online ASS problem to a solution for the online BST problem with exactly the same cost. However, this transformation is possible if we allow the cost of the BST algorithm to be at most a constant multiple of the ASS algorithm's cost [4].

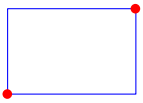

Fig. 1. An unsatisfied pair of points. The closed axis-aligned rectangle with corners defined by the pair is shown.

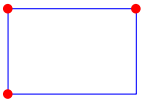

Fig. 2. An arborally satisfied superset of the same pair of points

3 GreedyFuture

We now turn our focus to describing the GreedyFuture algorithm in more detail. Let $S = \langle s_1, \ldots, s_m \rangle$ be an arbitrary search sequence of length m. After every search, GreedyFuture will rearrange the search path to minimize the cost of future searches.

More precisely, consider the ith search for the given sequence S. If $i = m$, then GreedyFuture does not rearrange the search path. Otherwise, if s_{i+1} lies on the search path τ_i, GreedyFuture makes s_{i+1} the root of τ_i'. If s_{i+1} does not lie along the search path, then GreedyFuture makes the predecessor and successor of s_{i+1} *within* τ_i the root and root's right child of τ_i' (if the successor (predecessor) does not exist, then GreedyFuture makes the predecessor (successor) the root and does not assign a right (left) child within τ_i'.) Now that it has fixed one or two nodes x_ℓ and x_r with $x_\ell < x_r$, GreedyFuture recursively sets the remaining nodes of τ_i *less* than x_ℓ using the subsequence of $\langle s_{i+1}, \ldots, s_m \rangle$ containing nodes less than x_ℓ. It then sets the nodes of τ_i greater than x_r using the subsequence of $\langle s_{i+1}, \ldots, s_m \rangle$ containing nodes greater than x_r.

Taking a cue from Demaine et al., we will call the online geometric model of the algorithm GreedyASS. Let $X = P(S)$ for some BST access sequence S. At each time i, GreedyASS simply outputs the minimal set of points at $y = i$ needed to satisfy X up to $y \leq i$.

We note that the set of points needed to satisfy X up to $y \leq i$ is uniquely defined. For each unsatisfied rectangle formed with (s_i, i) in one corner, we add the other corner at $y = i$. We can also define GreedyASS as an algorithm that sweeps right and left from the search node, accessing nodes that have increasingly greater last access times. See Fig. 3.

GreedyASS, the online geometric view of GreedyFuture, greatly reduces the complexity of predicting GreedyFuture's behavior. By focusing our attention on this geometric algorithm, we proceed to prove several upper bounds on both algorithms' performance in the following section.

4 An Access Lemma and Its Corollaries

In their paper on splay trees, Sleator and Tarjan prove the *access lemma*, a very general expression detailing the amortized cost of a splay (and therefore search) operation [12]. They use this lemma to prove several upper bounds, including the entropy bound, the static finger bound, and the working set bound.

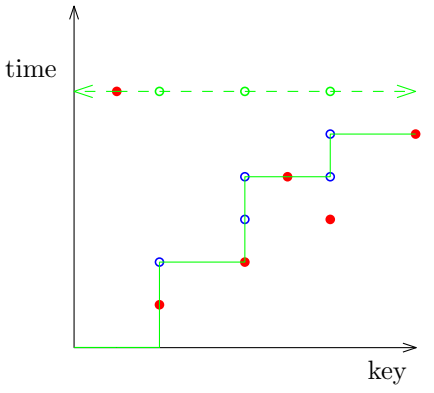

Fig. 3. (Left) A sample execution of GreedyASS. Search elements are represented as solid disks. For the latest search, GreedyASS sweeps right, placing points when the greatest last access time seen increases. The staircase represents these increasing last access times.

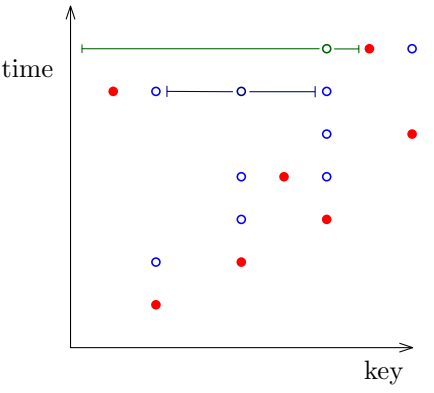

Fig. 4. (Right) Later in the same execution of GreedyASS. The most recent neighborhoods for two of the elements are represented as line segments surrounding those elements. Observe that adding another search for anything within a neighborhood will result in accessing the corresponding element for that neighborhood.

Wang et al. prove a similar lemma for their multi-splay tree data structure to show $O(\log\log n)$-competitiveness and $O(\log n)$ amortized performance, and the version of the lemma given in Wang's Ph.D. thesis is used to prove the other distribution sensitive upper bounds listed above [15,14]. In this section, we provide a similar lemma for GreedyASS and discuss its consequences.

4.1 Potentials and Neighborhoods

Fix a BST access sequence S and let $X = P(S)$. We consider the execution of GreedyASS on X. Let $\rho(x,i)$ be the last access of x at or before time i. Formally, $\rho(x,i)$ is the y coordinate of the highest point on the closed ray from (x,i) to $(x,-\infty)$.

Let a be the greatest positive integer smaller than x such that $\rho(a,i) \geq \rho(x,i)$ (or let $a = 0$ if no such integer exists). The ***left neighborhood*** of x at time i is $\{a+1, a+2, \ldots, x-1\}$ and denoted $\Gamma_\ell(x,i)$. The ***right neighborhood*** of x at time i is defined similarly and denoted $\Gamma_r(x,i)$. Finally, the ***inclusive neighborhood*** of x at time i is $\Gamma(x,i) = \Gamma_\ell(x,i) \cup \Gamma_r(x,i) \cup \{x\}$.

The inclusive neighborhood of x at time i contains precisely those keys whose appearance as s_{i+1} would prompt GreedyASS to access x at time $i+1$. Intuitively, the inclusive neighborhood is similar to a node's subtree in the arboral model. See Fig. 4.

Assign to each element $x \in \{1, \ldots, n\}$ a positive real weight $w(x)$. The ***size*** of x at time i is $\sigma(x,i) = \sum_{e \in \Gamma(x,i)} w(e)$. The ***rank*** of x at time i

is $r(x,i) = \lfloor \lg \sigma(x,i) \rfloor$. Finally, define a potential function $\Phi(i) = \sum_{x \in [n]} r(x,i)$ and let the ***amortized cost*** of a search at time i be $1 + |P_i| + \Phi(i) - \Phi(i-1)$.

Lemma 1 (Access Lemma). *Let $W = \sum_{x \in [n]} w(x)$. The amortized cost of a search at time i is at most $5 + 6 \lfloor \lg W \rfloor - 6r(s_i, i-1)$.*

4.2 Immediate Consequences

Before we proceed to prove Lemma 1, we will show several of its consequences. Recall that the equivalence between the arboral and geometric models mean these corollaries apply to both GreedyASS and GreedyFuture. The proofs of these corollaries mirror the proofs by Sleator and Tarjan for splay trees [12].

Corollary 2 (Balance Theorem). *The total cost of searching is $O((m+n) \times \log n)$.*

Corollary 3 (Static Optimality Theorem). *Let $t(x)$ be the number of times x appears in the search sequence S. If every element is searched at least once, the total cost of searching is $O\left(m + \sum_{x=1}^{n} t(x) \log\left(m/t(x)\right)\right)$.*

Corollary 4 (Static Finger Theorem). *Fix some element f. The total cost of searching is $O(m + \sum_{i=1}^{m} \log(|s_i - f| + 1))$.*

Corollary 5 (Working Set Theorem). *Let $d(i)$ be the number of distinct elements in the search sequence S before s_i and since the last instance of s_i. If there are no earlier instances of s_i, then let $d(i) = i - 1$. The total cost of searching is $O(m + \sum_{i=1}^{m} \log(d(i) + 1))$.*

Note that Corollary 5 implies other upper bounds on GreedyFuture's performance such as key-independent optimality [9].

4.3 Telescoping Rank Changes

We proceed to prove Lemma 1. First we observe the following.

Lemma 6. *Let x be any element* not *accessed during search i. Then we have $\Gamma(x, i-1) = \Gamma(x,i)$.*

Proof: Assume without loss of generality that $x > s_i$. Let x_ℓ be the greatest element in $\{s_i, s_i + 1, \ldots, x-1\}$ such that $\rho(x_\ell, i-1) \geq \rho(x, i-1)$. Element x_ℓ must exist, because GreedyASS does not access x at time i. No elements in $\{x_\ell + 1, \ldots, x-1\}$ are accessed at time i since they have smaller last access time than x_ℓ, so $\Gamma_\ell(x, i-1) = \Gamma_\ell(x,i)$. Likewise, no elements in $\Gamma_r(x, i-1)$ are accessed at time i since they have smaller last access time than x. The inclusive neighborhood of x (as well as its size and rank) remains unchanged by the search. □

Consider a search at time i. Lemma 6 immediately implies the amortized cost of the search is equal to

$$\sum_{x \in P_i \cup \{s_i\}} (1 + r(x,i) - r(x,i-1)) . \tag{1}$$

Suppose we access an element $x \neq s_i$. Assume $x > s_i$ without loss of generality. If it exists, let x_r be the least accessed element greater than x. We call x_r the ***successor*** of x. Observe that $\Gamma(x,i)$ contains a subset of the elements in $\{s_i + 1, \ldots, x_r - 1\}$ while $\Gamma(x_r, i-1)$ contains a superset of the elements in $\{s_i, \ldots, x_r\}$. This fact implies $\Gamma(x,i) \subset \Gamma(x_r, i-1)$ which in turn implies

$$\sigma(x,i) < \sigma(x_r, i-1) \text{ and } r(x,i) \leq r(x_r, i-1). \tag{2}$$

If the second inequality is strict, then

$$1 + r(x,i) - r(x,i-1) \leq r(x_r, i-1) - r(x, i-1). \tag{3}$$

Otherwise,

$$1 + r(x,i) - r(x,i-1) = 1 + r(x_r, i-1) - r(x, i-1). \tag{4}$$

Call an accessed element $x > s_i$ a ***stubborn element*** if x has a successor x_r and $r(x,i) = r(x_r, i-1)$. From (1), (3), and (4) above, the amortized cost of accessing elements greater than s_i forms a telescoping sum and we derive the following lemma.

Lemma 7. *Let α be the number of elements greater than s_i that are stubborn and let $e_{r\ell}$ and e_{rr} be the least and greatest elements greater than s_i to be accessed. The amortized cost of accessing elements greater than s_i is*

$$1 + \alpha + r(e_{rr}, i) - r(e_{r\ell}, i-1).$$

4.4 Counting Stubborn Elements

The biggest technical challenge remaining is to upper bound the number of stubborn elements α. We have the following lemma.

Lemma 8. *The number of accessed elements greater than s_i which are stubborn is at most*

$$1 + 2\lfloor \lg W \rfloor - 2r(s_i, i-1)$$

Proof: Consider any stubborn element $x > s_i$ and its successor x_r. Let the ***left size*** of x at time i be $\sigma_\ell(x,i) = \sum_{e \in \Gamma_\ell(x,i)} w(e)$. Further, let the ***left rank*** of x at time i be $r_\ell(x,i) = \lfloor \lg(\sigma_\ell(x,i)) \rfloor$. By the definitions of stubborn elements and left sizes we see

$$\sigma(x,i) > \frac{1}{2}\sigma(x_r, i-1) > \frac{1}{2}\sigma_\ell(x_r, i-1). \tag{5}$$

We note that for any accessed element v (stubborn or not) with $s_i < v < x$ we have

$$\sigma_\ell(v, i-1) < \frac{1}{2}\sigma_\ell(x_r, i-1) \tag{6}$$

by (5) since every element of $\Gamma_\ell(v, i-1)$ is in $\Gamma_\ell(x_r, i-1)$, but none of these elements are in $\Gamma(x, i)$ since the left neighborhood of x at time i cannot extend past v. Further,

$$\sigma_\ell(x, i-1) \geq \sigma(s_i, i-1) \tag{7}$$

since all weights are positive and every element in $\Gamma(s_i, i-1)$ is also in $\Gamma_\ell(x, i-1)$.

Let $z > s_i$ be the greatest stubborn element, and let z_r be its successor. We will inductively argue the number of stubborn elements is at most

$$1 + 2r_\ell(z_r, i-1) - 2r(s_i, i-1)$$

which is a stronger statement than that given in the lemma. The argument can be divided into two cases.

1. Suppose $\sigma_\ell(z_r, i-1) < 2\sigma(s_i, i-1)$. For any stubborn element v between s_i and z we have
$$\sigma_\ell(v, i-1) < \sigma(s_i, i-1)$$
by (6). There can be no such element v by (7), making z the only stubborn element. The total number of stubborn elements is
$$\begin{aligned} 1 &\leq 1 + 2r_\ell(z, i-1) - 2r(s_i, i-1) \\ &\leq 1 + 2r_\ell(z_r, i-1) - 2r(s_i, i-1) \end{aligned}$$
by (7) and the definition of left rank.
2. Now suppose $\sigma_\ell(z_r, i-1) \geq 2\sigma(s_i, i-1)$. Consider any stubborn element v with successor v_r such that $s_i < v < v_r < z$. Note that if a stubborn element exists with z as its successor, v cannot be this stubborn element. We have
$$\sigma_\ell(v_r, i-1) < \frac{1}{2}\sigma_\ell(z_r, i-1)$$
by (6). By induction on the left sizes of stubborn element successors greater than s_i, the successors of at most
$$1 + 2\left\lfloor \lg\left(\frac{1}{2}\sigma_\ell(z_r, i-1)\right)\right\rfloor - 2r(s_i, i-1)$$
stubborn elements can have this smaller left size. Counting z and the one other stubborn element that may exist with z as its successor, the total number of stubborn elements is at most
$$3 + 2\left\lfloor \lg\left(\frac{1}{2}\sigma_\ell(z_r, i-1)\right)\right\rfloor - 2r(s_i, i-1) = 1 + 2r_\ell(z_r, i-1) - 2r(s_i, i-1).$$

□

4.5 Finishing the Proof

We now conclude the proof of Lemma 1.

Proof: By Lemma 6, the amortized cost of accessing s_i alone is

$$1 + r(s_i, i) - r(s_i, i-1) \leq 5 + 6\lfloor \lg W \rfloor - 6r(s_i, i-1)$$

so the lemma holds in this case.

If all other accessed elements are greater than s_i, let $e_{r\ell}$ and e_{rr} be the least and greatest of these elements. Observe $r(e_{r\ell}, i-1) \geq r(s_i, i)$ and $r(e_{rr}, i) \leq \lfloor \lg W \rfloor$. By Lemma 7 and Lemma 8, the total amortized cost of accessing elements is at most

$$\begin{aligned} &3 + r(s_i, i) - 3r(s_i, i-1) + 2\lfloor \lg W \rfloor + r(e_{rr}, i) - r(e_{r\ell}, i-1) \\ &\quad \leq 3 + 3\lfloor \lg W \rfloor - 3r(s_i, i-1) \\ &\quad \leq 5 + 6\lfloor \lg W \rfloor - 6r(s_i, i-1) \end{aligned}$$

so the lemma holds in this case. It also holds in the symmetric case when all accessed elements are smaller than s_i.

Finally, consider the case when there are accessed elements both greater than and less than s_i. Let $e_{\ell\ell}$ and $e_{\ell r}$ be the least and greatest elements *less than* s_i. Observe $r(e_{\ell\ell}, i) \leq \lfloor \lg W \rfloor$ and $r(e_{\ell r}, i-1) \geq r(s_i, i-1)$. By two applications of Lemma 7 and Lemma 8, the total amortized cost of the search is at most

$$\begin{aligned} &5 + r(s_i, i) - 5r(s_i, i-1) + 4\lfloor \lg W \rfloor + r(e_{rr}, i) - r(e_{r\ell}, i-1) \\ &\qquad + r(e_{\ell\ell}, i) - r(e_{\ell r}, i-1) \\ &\quad \leq 5 + 6\lfloor \lg W \rfloor - 6r(s_i, i-1) \end{aligned}$$

□

5 A Sequential Access Theorem

The working set bound proven above shows that GreedyFuture has good *temporal locality*. Accessing an element shortly after its last access guarantees a small amortized search time. Sleator and Tarjan conjectured that their splay trees also demonstrate good spatial locality properties in the form of the dynamic finger conjecture [12]. This conjecture was verified by Cole, et al. [3,2].

One special case of the dynamic finger theorem considered by Tarjan and others was the sequential access theorem [13,7,15,14]. We give a straightforward proof of the sequential access theorem when applied to GreedyFuture. Note that this theorem requires focusing on an arbitrary fixed BST, so we do not use the geometric model in the proof.

Theorem 9 (Sequential Access Theorem). *Let $S = \langle 1, 2, \ldots, n \rangle$. Starting with an arbitrary BST T_0, the cost of running* GreedyFuture *on search sequence S is $O(n)$.*

Let $T_0, T_1, \ldots, T_n$ be the sequence of search trees configured by GreedyFuture. We make the following observations:

Lemma 10. *For all $i > 1$, either node i is the root of T_{i-1} or $i-1$ is the root and i is the leftmost node of the root's right subtree.*

Proof: If i was accessed during the $i-1$st search, then i is the root of T_{i-1}. Otherwise, $i-1$ is the predecessor node of i on the search path. Therefore, $i-1$ is the root of T_{i-1} and i is the leftmost node of the root's right subtree. □

Lemma 11. *Node x is accessed at most once in any position other than the root or the root's right child.*

Proof: Consider node x and search i. Node x cannot be accessed if $x < i-1$ according to Lemma 10. If x lies on the search path and $x \leq i+1$ then either x becomes the root or x moves into the root's left subtree so that i or $i+1$ can become the root.

Now suppose x lies along the search path and $x > i+1$. Let x_ℓ be the least node strictly smaller than x that does not become the root. If x_ℓ does not exist, then x becomes the root's right child as either x is the successor of $i+1$ on the search path, node $i+1$ is on the search path and $x = i+2$, or node $i+1$ is on the search path and x is the successor of $i+2$ on the search path. If x_ℓ does exist, then x_ℓ becomes the root's right child for one of the reasons listed above and x becomes a right descendent of x_ℓ.

Node x cannot be moved to the left subtree of the root's right child in all the cases above. Lemma 10 therefore implies x is accessed in the root's left subtree on the first search, x is accessed *once* in the left subtree of the root's right child, or x is never accessed anywhere other than as the root or root's right child. □

We now conclude the proof of Theorem 9.

Proof: The cost of the first search is at most n. The costs of all subsequent searches is at most $2(n-1)+n$ according to Lemma 11; at most $2(n-1)$ node accesses occur at the root or root's right child, and at most n nodes are accessed exactly once in a position other than the root or the root's right child. The total cost of all searches is at most $4n-2$. □

6 Closing Remarks

The ultimate goal of this line of research is to prove GreedyFuture or splay trees optimal, but showing other upper bounds may prove interesting. In particular, it would be interesting to see if some difficult to prove splay tree properties such as the dynamic finger bound have concise proofs when applied to GreedyFuture. Another direction is to explore how GreedyFuture may be modified to support insertions and deletions while still maintaining its small search cost.

Acknowledgements. The author would like to thank Alina Ene, Jeff Erickson, Benjamin Moseley, and Benjamin Raichel for their advice and helpful discussions as well as the anonymous reviewers for their suggestions on improving this report.

This research is supported in part by the Department of Energy Office of Science Graduate Fellowship Program (DOE SCGF), made possible in part by the American Recovery and Reinvestment Act of 2009, administered by ORISE-ORAU under contract no. DE-AC05-06OR23100.

References

1. Bose, P., Douïeb, K., Dujmović, V., Fagerberg, R.: An $O(\log \log n)$-competitive binary search tree with optimal worst-case access times. In: Kaplan, H. (ed.) SWAT 2010. LNCS, vol. 6139, pp. 38–49. Springer, Heidelberg (2010)
2. Cole, R.: On the dynamic finger conjecture for splay trees. Part II: The proof. SIAM J. Comput. 30, 44–85 (2000)
3. Cole, R., Mishra, B., Schmidt, J., Siegel, A.: On the dynamic finger conjecture for splay trees. Part I: Splay sorting $\log n$-block sequences. SIAM J. Comput. 30, 1–43 (2000)
4. Demaine, E.D., Harmon, D., Iacono, J., Kane, D., Pătraşcu, M.: The geometry of binary search trees. In: Proc. 20th ACM/SIAM Symposium on Discrete Algorithms, pp. 496–505 (2009)
5. Demaine, E.D., Harmon, D., Iacono, J., Pătraşcu, M.: Dynamic optimality–almost. SIAM J. Comput. 37(1), 240–251 (2007)
6. Derryberry, J., Sleator, D.D., Wang, C.C.: A lower bound framework for binary search trees with rotations. Tech. Rep. CMU-CS-05-187. Carnegie Mellon University (2005)
7. Elmasry, A.: On the sequential access theorem and deque conjecture for splay trees. Theoretical Computer Science 314(3), 459–466 (2004)
8. Goyal, N., Gupta, M.: On dynamic optimality for binary search trees (2011), http://arxiv.org/abs/1102.4523
9. Iacono, J.: Key independent optimality. Algorithmica 42, 3–10 (2005)
10. Lucas, J.M.: Canonical forms for competitive binary search tree algorithms. Tech. Rep. DCS-TR-250. Rutgers University (1988)
11. Munro, J.I.: On the competitiveness of linear search. In: Paterson, M. (ed.) ESA 2000. LNCS, vol. 1879, pp. 338–345. Springer, Heidelberg (2000)
12. Sleator, D.D., Tarjan, R.E.: Self-adjusting binary search trees. Journal of the Association for Computing Machinery 32(3), 652–686 (1985)
13. Tarjan, R.E.: Sequential access in splay trees takes linear time. Combinatorica 5, 367–378 (1985)
14. Wang, C.C.: Multi-Splay Trees. Ph.D. thesis. Carnegie Mellon University (2006)
15. Wang, C.C., Derryberry, J., Sleator, D.D.: $O(\log \log n)$-competitive binary search trees. In: Proc. 17th Ann. ACM-SIAM Symp. Discrete Algorithms, pp. 374–383 (2006)
16. Wilber, R.E.: Lower bounds for accessing binary search trees with rotations. SIAM J. Comput. 18(1), 56–67 (1989)