1 Overview

In the last lecture we discussed splay trees, as detailed in [ST85]. We saw that splay trees are $c$-competitive with the optimal binary search tree in the static case by analyzing the 3 operations used by splay trees.

In this lecture we continue our analysis of splay trees with the introduction and proof of several properties that splay trees exhibit and introduce the geometric view of BSTs. In Section 2 we define several properties of splay trees and BSTs in general. In Section 3 we introduce the geometric view of BSTs by defining arborally satisfied sets of points and proving that they are equivalent to BSTs.

2 Splay Tree Properties

Property 1. (Static Finger Bound): The amortized cost $\hat{c}_f(x)$ of searching for $x$ in a splay tree, given a fixed node $f$ is:

$$\hat{c}_f(x) = O(\log (1 + |x - f|))$$

where $|x - f|$ is the distance from node $x$ to $f$ in the total order.

The Static Finger Bound says that accessing any element $x$ can be bounded by the logarithm of the distance from some designated node $f$. Informally, it can be viewed as placing a finger on node $f$, starting each search from $f$ in a tree balanced around $f$.

Proof. To prove this property for splay trees, let the weight function be $w(x) = \frac{1}{(1 + x - f)^2}$. Then $s_{root}$, the value of the root node, can be bounded by:

$$s_{root} = \sum_{x \in \text{tree}} \frac{1}{(1 + x - f)^2}$$

$$\leq \sum_{k=-\infty}^{\infty} \frac{1}{(1 + k - f)^2}$$

$$\leq \sum_{-\infty}^{\infty} \frac{1}{k^2} = \frac{\pi^2}{3}$$

$$= O(1)$$
This tells us that the size function of the root node is bounded by a constant. From the previous lecture, we know that the amortized cost of searching for \( x \), \( \hat{c}(x) \) is:

\[
\hat{c}(x) = O \left( \log \frac{s_{\text{root}}}{s_0(x)} \right)
= O \left( \log \frac{O(1)}{1 + x - f} \right)
= O(\log (1 + |x - f|))
\]

Property 2. (Sequential access) [ST85]: Accessing \( n \) elements in sorted order in a splay tree takes \( O(n) \) time \( O(1) \) amortized per element).

Intuitively, splay trees have sequential access property because the splay operation moves elements close to the search target node near the top of the tree and searching for next element are quick. This property captures spatial locality of data access. Spatial locality is a property of data access that provides improved access time of elements that are spatially near to previously accessed elements. Many cache systems implement spatial locality by caching a block of memory surrounding an accessed element.

The sequential access property follows from the following more general property of splay trees.

Property 3. (Dynamic finger) [Col00a, Col00b]: After accessing element \( x \), the amortized cost of accessing \( y \) in splay trees is \( O(\log (1 + |x - y|)) \).

Informally, dynamic finger property says that when performing a search for key \( x \), we remember which node we found it in by moving our finger to it. The next search (for key \( y \)) will start from that node and the time to find \( y \) will be proportional to the distance from \( x \) to \( y \) in a tree balanced around \( x \).

The proof of this property is very long and complex and is beyond the scope of this class. It has been recently simplified and expanded [IL16].

Property 4. (Working set) [ST85]: The amortized cost of searching for \( x \) in a splay tree is \( O(1 + \log t_x) \), where \( t_x \) is the number of distinct elements searched for since the last search for \( x \).

The working set property captures temporal locality of data access. Temporal locality is a property of data access where elements that are recently accessed are likely to be accessed soon again. The least recently used (LRU) cache replacement policy in modern CPU caches take advantage of temporal locality of accesses because the most recently accessed elements are cached and can be accessed more quickly.

Property 5. (Unified property) (not proven for splay trees): If \( t_{ij} \) is the number of distinct elements between searches for \( x_i \) and \( x_j \), then a BST has unified property if the cost for searching \( x_j \) is \( O(\log (\min_{i<j} (|x_i - x_j| + t_{ij} + 2))) \).
While unified property is conjectured to be true for splay trees, it has yet to be proven and is an important open problem. Note that unified property combines both the dynamic finger property and the working set property. And a BST with the unified property will have the other two as well. While it is conjectured that splay trees have unified property, [Iac01] has proven that:

Theorem 6. There exists a BST with Unified property [Iac01].

2.1 Model Definitions

As an aside, in this lecture we saw the following broad data structure models:

Definition 7. Pointer model: Any structure that can be represented by nodes with pointers. An algorithm can remember $O(1)$ nodes and is allowed to follow from node to node along the edges and modify/redirect pointers to nodes in its local memory/storage. This is a broad category of data structure, since there are no restrictions placed on the number of pointers per node or modifications that can be made to pointers.

Definition 8. BST model: A structure of nodes and pointers where:

- every search starts at the root node
- left or right child can be seen from each node
- parent can be visited from each node
- rotation operation can be performed

Note that the BST model is a more restrictive model of data structures. Note that splay trees can be implemented in the BST model.

2.2 Dynamic Optimality

Definition 9. A BST is dynamically optimal if it is $O(1)$-competitive with any other algorithm in the BST model on every search sequence $X$ of size $n$, where $n$ is asymptotically large.

It is conjectured that splay trees are dynamically optimal and would be a very significant result if this were proven. However, currently the closest result are structures that are shown to be $O(\log \log n)$-competitive with the optimal algorithm for every sequence $X$. The following data structures have been proven to be $O(\log \log n)$-competitive with the optimal algorithm in the BST model:

- Tango Trees [DHIP04] - While $O(\log \log n)$-competitive with the optimal, they require $O(\log n \cdot \log \log n)$ time per search in the worst case.
- Multi-Splay Trees [WDS06] - Improve on Tango Trees by performing search queries in $O(\log n)$ amortized time, while remaining $O(\log \log n)$-competitive on a sequence on $n$ queries. However, in the worst-case they require $O(\log^2 n)$ per search.
- Zipper Trees [BDDF10] - The best solution yet with an optimal $O(\log n)$ worst case and is proven to be $O(\log \log n)$-competitive with the optimal algorithm for a sequence of $n$ queries.
3 Geometric View of BSTs

The geometric view is a way of visualizing BSTs and searches over BSTs as points in the Euclidean plane.

In the geometric view of BSTs, we consider a 2-dimensional plane where the $x$-axis represents the search keys. The $y$-axis corresponds to the time at which the search queries happen in the BST. Each node that is accessed during a search will be plotted as follows: we place a point with coordinates $(x, i)$ for every node with key $x$ that has been touched during the $i$th search query in the BST. Figure 1 illustrates how a sequence of searches on a static BST are plotted in the geometric view.

While there is a geometric view for each BST and a sequence of queries, it is also true that for each geometric view there is a corresponding BST and a sequence of searches. To show this, we need the following definition.

**Definition 10.** A point set $P$ is **arborally satisfied** if for every pair of points $(x, i) \in P$ and $(y, j) \in P$ (w.l.o.g. assume $i \leq j$ and $x \leq y$ – the other cases are symmetric), one of the following is true:
1) $x = y$
2) $i = j$
3) $\exists (z, k) \in P$ s.t. $x \leq z \leq y$ and $i \leq k \leq j$

Informally, this says that for a set of points to be arborally satisfied, every pair of points must either be on the same horizontal or vertical line, or there must be another point within (including the boundaries) the rectangle formed by those two points as corners.

**Lemma 11.** In an arborally satisfied set (ASS) $P$, for every rectangle formed by a pair of points $p \in P$ and $q \in P$ that are not on the same horizontal or vertical line, there is a point on a rectangle side incident on $p$ (that is not $p$) and a point on a rectangle side incident on $q$ (that is not $q$).
Figure 2: An illustration of the proof that an arborally satisfied set of points must have a point on incident edges.

Proof. We will prove that there is a point on a rectangle side incident on \( p \). Proof for \( q \) is symmetric.

Since \( P \) is ASS, there must be a point \( r \in P \) inside the rectangle formed by \( p \) and \( q \), as seen in Figure 2. If \( r \) is on a side incident on \( p \) we are done. Otherwise, consider the rectangle between \( r \) and \( p \) and repeat the process. We continue this process, until we either find a point that is on a side incident on \( p \) or we find a rectangle that is not arborally satisfied, which is a contradiction to the fact that \( P \) is ASS. Therefore, there must be a point on a side incident on \( p \).

\[ \text{Theorem 12. A point set } P = \{(x,i)\} \text{ is arborally satisfied iff it corresponds to a valid BST with a sequence of searches that touch node with key } x \text{ during the } i\text{-th search query.} \]

This theorem states that any ASS represents a valid BST and every valid BST with a set of searches can be represented by an ASS. We saw a simple example of this in Figure 1 though this was for a simple static BST with no rotation operations. However, this theorem tells us that an ASS can represent any algorithm within the BST model, including splay trees. We will prove this theorem in the next lecture.

References


