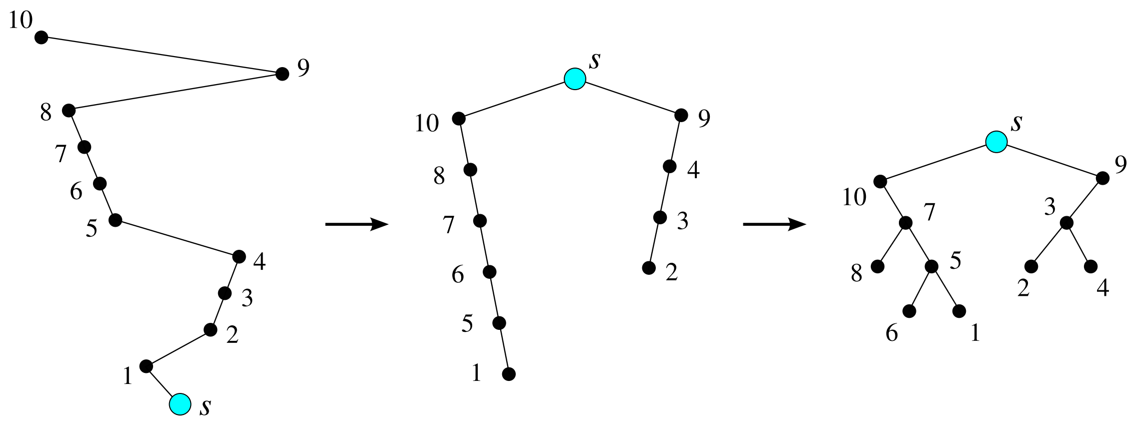
A New Path from Splay to Dynamic Optimality by Caleb Levy and Bob Tarjan (SODA 2019)

Reading Group Talk by Kurt Mehlhorn

Oct. 24, 2018

1 Introduction and Statement of Main Results

The paper revisits the dynamic optimality conjecture for splay trees. Splay trees were introduced by Sleator and Tarjan in ’84. Splay trees support searches, insertions, deletions, joins, splits, ... In this talk, we are concerned with sequences of searches. Splay rebuilds the tree after every search as illustrated in the following figure taken from [CGK+15].



The search path to the accessed element s is rebuilt as follows. Number the nodes on the search path to s bottom up, starting at zero. Make s the root and attach to the root the two paths of smaller and larger nodes respectively. Then on both paths perform the following rotations. If the nodes numbered $2k+1$ and $2k+2$ end up in the same path, perform a rotation on the edge connecting them. In the example, we perform rotations on the edges $(5,6)$, $(7,8)$, and $(3,4)$. This description of splay is a global view of splay introduced in [CGK+15]. The standard description is a bottom-up rebuild of the search path.

The cost of a search (the cost of serving a request) is the length of the search path and the cost of a sequence of searches is the sum of the costs of the individual searches. We use $Cost(T,X)$ to denote the cost of serving X with initial tree T by splay.

Conjecture 1 (Dynamic Optimality) *There is a constant c such that*

$$Cost(T,X) \le c \cdot Opt(T,X)$$

for every initial tree T and request sequence X.

We still need to define $Opt(T,X)$. A search tree algorithm serves requests by subtree replacements. Let T' be the search tree before serving the request to an item x. The algorithm selects a subtree Q containing the root of T' and x and replaces Q by a tree Q' on the same set of nodes and rooted at x. The dangling subtrees are attached in the unique way so at to maintain the search tree property. The cost is the size of Q. For a sequence, we again sum up the costs of the individual operations. $Opt(T,X)$ is the minimum cost of serving X starting with T.

In [CGK$^+$15] it was shown that subtrees in which no node is every accessed can be ignored, i.e., deleted. Then, at least initially, every node lies on a path to an accessed element and hence $Opt(T,X) \geq |T|$. Also, the cost of each access is at least one and hence $Opt(T,X) \geq |X|$.

Note that *Opt* is an offline algorithm and splay is an online algorithm.

A subsequence Y of a sequence X of requests is obtained by deleting requests from X.

Conjecture 2 (Subsequence Property) *There is a constant c such that for any sequence X of requests, initial tree T, and any subsequence Y of X,*

$$Cost(T,Y) \leq c \cdot Cost(T,X).$$

Theorem 1 *Splay satisfies dynamic optimality iff splay has the subsequence property.*

Some researchers suggested that one should weaken the dynamic optimality conjecture by adding an additive term that depends only on the size of the initial tree. After all, Opt starting in any tree can first convert this tree into its most favorite tree. However, splay may have to pay a warm-up cost.

Conjecture 3 (Dynamic Optimality with Warm-Up Cost) *There is a constant c and a function g such that*

$$Cost(T,X) \leq c \cdot Opt(T,X) + g(|T|)$$

for every initial tree T and request sequence X.

Theorem 2 *Splay satisfies dynamic optimality iff splay satisfies dynamic optimality with a warm-up cost.*

We now come to the key technical insight of the paper. In a certain sense, splay can simulate any other search tree algorithm.

Theorem 3 (Splay has the Simulation Property) *For any search tree algorithm B there is a constant c such that for every initial tree T and every sequence X of requests there is a sequence X_B of requests such that*
(a) X is a subsequence of X_B,
(b) $Cost(T,X_B) \leq c \cdot Cost_B(T,X)$, where $Cost_B$ is the cost of processing X by B starting with T.

Theorem 4 *Subsequence Property + Simulation Property $\Rightarrow$ Dynamic Optimality.*

Theorem 5 *Dynamic Optimality $\Rightarrow$ Subsequence Property*

2 Proofs

Proof: Dynamic Optimality $\Rightarrow$ Subsequence Property: Let T be a tree, X be a sequence of requests, and Y a subsequence of X. Then

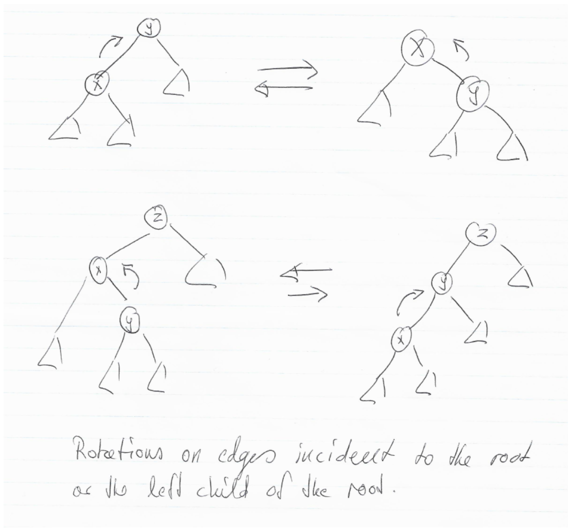
$$Cost(T,Y) \leq c \cdot Opt(T,Y) \leq c \cdot Opt(T,X) \leq c \cdot Cost(T,X),$$

where the first inequality follows from the dynamic optimality of splay, the second inequality holds since Opt has the subsequence property (with constant equal to 1), and the last inequality holds since the cost of Opt is no larger than the cost of splay on the same sequence.

Intuitively, Opt has the subsequence property because the sequence of actions serving X also serves Y. Technically, it requires to collapse sequences of subtree replacements into a single subtree replacement. ∎

Proof: Subsequence Property + Simulation Property $\Rightarrow$ Dynamic Optimality: Let T be any tree and let X be any sequence of requests. By the simulation property of splay, there is a supersequence Z of X such that $Cost(T,Z) \leq c \cdot Opt(T,X)$. By the subsequence property of splay, $Cost(T,X) \leq d \cdot Cost(T,Z)$. Thus $Cost(T,X) \leq c \cdot d \cdot Opt(T,X)$. ∎

We come to the simulation property. A *restricted rotation* is a rotation at an edge incident to the root or incident to the left child of the root. So there are 4 different restricted rotations. They come in pairs that are inverses of each other.



Rotations on edges incident to the root or the left child of the root.

Note that restricted rotations affect only tiny subtrees (two or three nodes) rooted at the root. The following Lemma was well-known.

Lemma 1 (Lucas, and others) *Any tree T_1 on n nodes can be transformed into any other tree T_2 on the same set of nodes by $4n$ restricted rotations.*

Proof: The idea is to transform T_1 into a left-leaning path (the root is node n and the leftmost leaf is node 1) and then the left-leaning path into T_2. The second transformation is the inverse of transforming T_2 into the left-leaning path. I prove a weaker bound of $8n$, $4n$ for each of the two parts.

In order to transform T_1 into a path, let v be the deepest node on the left spine which has a right child. There are four cases: v does not exist, v exists and is the root, v exists and is the left child of the root, v exists and has depth 2 or more. If v does not exist, we are done. If v exists and is the root, we rotate on the right edge incident to the root. This will turn the right child of the root into the root with the former root its left child. If v exists and is the left child of the root, we rotate on the right edge incident to v. This moves the right child of v onto the spine and makes v a grandchild of the root. If v exists and is not the left child of the root, we rotate on the left edge incident to the root. This will move v closer to the root.

How does one measure progress and prove linear time? We first observe that the rotations at edges incident to the root do not change the number of nodes lying on the two spines and the rotation on the right edge incident to the left child of the root increases it by one. Thus there at most n rotations of this kind; call it a type 1 rotation.

At the beginning we make at most n rotations at the left edge (type 2 rotation) incident to the root to make the deepest node on the left spine a child of the root. Lateron, we only make a type 2 rotation if after a type 1 rotation the node v is a grandchild of the root. Thus there are at most $n+n$ type 2 rotations.

A rotation at the right edge incident to the root (type 3 rotation) is only performed if the root is the only node on the left spine with a right child. Thus a type 3 rotation is preceded by a type 1 rotation. Thus there are most n type 3 rotations. ∎

Next comes the crucial insight. We consider the binary search trees on four nodes[1]. There are 14 such trees. We draw an edge from a tree T to a tree T' if there is a node x of T such that splay will transform T into T' after a search for x.

Lemma 2 *The transition graph on binary search trees of size 4 is strongly connected. It diameter is 5.*

Proof: The picture below (taken from Levy-Tarjan) shows that the graph has a Hamiltonian cycle.

[1]It is tempting to consider trees on three nodes as restricted rotations affect only three nodes. However, the transition graphs for trees of size three under splay it not strongly connected.

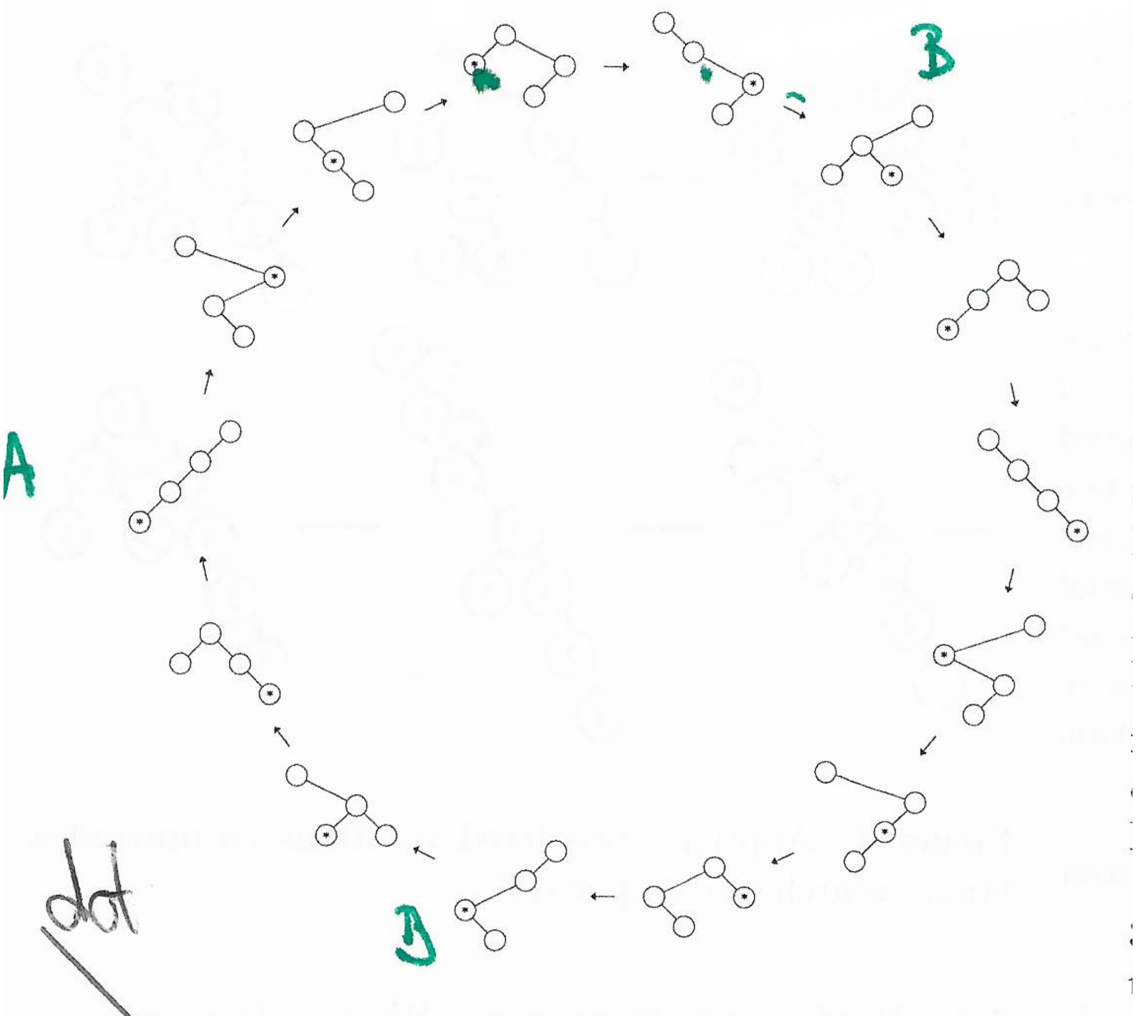


Figure 4: A Hamiltonian cycle in G_4. Splay at nodes marked by $*$ to transition from one tree to the next.

I skip the proof that the diameter is five. It is shown by an inspection of the transition graph. ∎

Example: Let us assume for concreteness that our trees of size 4 are on keys 1, 2, 3, and 4. In order to transform the tree A into the tree B in the figure above, we use the access sequence 1, 3, 2, 1, 3.

So a sequence of at most 5 accesses will induce splay to transform any tree on four nodes into any other tree on four nodes. Since the maximum length of a search path in a tree of four nodes is four, the total cost of these 5 accesses is at most 20. We are now ready for the simulation theorem.

Proof: Simulation Property of Splay Assume the algorithm B replaces T by T' after a search for x. Recall that T is a subtree containing the root of the before tree and the accessed element x and that T' has x as the root. We can transform T into T' by $4|T|$ restricted rotations. For each restricted rotation, there is an access sequence of length 5 which makes splay simulate the restricted rotation. Therefore, there is an access sequence A of length $20|T|$ which will make splay transform T into T'. The cost of processing this sequence by splay it at most $80|T|$. After a splay the accessed element becomes the root. Since x is the root of T', the last access in the sequence A is an access to x.

We make this replacement for every access. We obtain a supersequence Z of X of length at most $20 Cost_B(T,X)$ with $Cost(T,Z) \leq 4|Z| \leq 80 Cost_B(T,X)$. ∎

Proof: dynamic optimality after warm-up implies dynamic optimality Consider any sequence X of requests and let T be the initial tree. Processing X by splay results in a tree T'. There is an access sequence Y of length at most $20|T|$ that makes splay transform T' back to T. Let U be X followed by Y. Then $\mathit{Cost}(T,X) \le \mathit{Cost}(T,U)$ since X is a prefix of U. Let $k \star U$, be the access sequence U repeated k times. Then $\mathit{Cost}(T,k\star U) = k \cdot \mathit{Cost}(U)$ since the tree after processing U is again T. Together, we have

$$k \cdot \mathit{Cost}(T,X) \le k \cdot \mathit{Cost}(T,U) = \mathit{Cost}(T,k\star U) \le c \cdot \mathit{Opt}(T,k\star U) + g(|T|) \le (c+1)\cdot \mathit{Opt}(T,k\star U),$$

where the next to last inequality follows from dynamic optimality after warm-up and the last inequality holds only for sufficiently large k, namely when $g(|T|) \le |k\star U|$. Then $g(|T|) \le \mathit{Opt}(T,k\star U)$.

Let us next estimate the cost of Opt on $k\star U$. Every way of serving this sequence gives an upper bound on the cost of Opt. We first serve X in the optimal way for a cost of $\mathit{Opt}(T,X)$. Let T'' be the tree obtained. We then use $4|T''|$ restricted rotations (total cost at most $16|T''|$) to transform T'' into T'. Then we behave as splay and transform T' into T'' for a cost of most $80|T|$. We repeat k times. Thus

$$\mathit{Opt}(T,k\star U) \le k\cdot(\mathit{Opt}(T,X) + 96|T|) \le 97k\cdot \mathit{Opt}(T,X),$$

where the last inequality uses $|T| \le \mathit{Opt}(T,X)$.

Stringing our inequalities together, we obtain

$$k\cdot \mathit{Cost}(T,X) \le (c+1)\cdot \mathit{Opt}(T,k\star U) \le (c+1)\cdot 97 \cdot k \cdot \mathit{Opt}(T,X)$$

for all sufficiently large k. Dividing by k establishes dynamic optimality.

∎

3 Further Results

They show that the deque and the transversal conjecture follow from dynamic optimality.

They show that the Wilber2 lower bound satisfies the subsequence property. Wilber2 is a lower bound for the cost of any binary search tree serving a sequence of accesses X. The authors prove that Wilber2 has the subsequence property. The proof is involved. The paper only contains a sketch; already the sketch spreads over three pages. In the geometric view of binary search trees, the claim has a simple proof.

In the geometric view, the definition of Wilber2 is as follows. I quote from Demaine et al. "Consider each point p of X in increasing order by y coordinate. Consider the orthogonal envelope of all points in the quadrant (x, y) | x < p.x, y < p.y, and the orthogonal envelope of all points in the quadrant (x, y) | x > p.x, y < p.y. (The orthogonal envelope is the set of points that are not orthogonally dominated by any other point, when looking from p.) Merge the two envelopes and sort them by y coordinate. Now draw a rectangle between any pair of consecutive points switching between the left and right quadrants. It is easy to see that these rectangle are independent."

Lemma 3 *Let X be a point set and let q be a point not in X. Points have pairwise distinct x- and y-coordinates. Then* $\text{Wilber}(S) \le \text{Wilber}(S \cup q)$.

Proof: Let p, ℓ, r be three points in S such that the alternation (ℓ, r) is counted by Wilber2 for p. Also assume $\ell.x < p.x < r.x$.

We first show that the rectangle $R = (\ell.x, r.x) \times (\min(\ell.y, r.y), p.y)$ contains no point in S. Since ℓ and r are not dominated by any point in S when looking from p, the rectangle with corners p and ℓ and the rectangle with corners p and r does not contain any point in S. Furthermore, ℓ and r are consecutive points in y-order in the two envelopes. Therefore the rectangle $(\ell.x, r.x) \times (\min(\ell.y, r.y), \max(\ell.y, r.y))$ contains no point in S.

Consider now the point q. If q does not lie in R, then the alternation (ℓ, r) is still counted for p. So assume that $q \in R$. If $q.y > \max(\ell.y, r.y)$, the alternation (ℓ, r) is counted for q. If $\min(\ell.y, r.y) < q.y < \max(\ell.y, r.y))$ (recall that coordinates are distinct), and $q.x < p.x$, then (q, r) is counted for p. If $q.x > p.x$, (ℓ, q) is counted for p. ∎

In the sequence part of the paper, they sketch an (so far unsuccessful) approach to dynamic optimality.

References

[CGK$^+$15] Parinya Chalermsook, Mayank Goswami, László Kozma, Kurt Mehlhorn, and Thatchaphol Saranurak. Self-Adjusting Binary Search Trees: What Makes Them Tick?. In *Algorithms – ESA 2015*, volume 9294 of *Lecture Notes in Computer Science*, pages 300–312. Springer Berlin Heidelberg, 2015.