ELSEVIER

Available online at www.sciencedirect.com

SCIENCE DIRECT®

Theoretical Computer Science 314 (2004) 459–466

Theoretical Computer Science

www.elsevier.com/locate/tcs

Note

On the sequential access theorem and deque conjecture for splay trees

Amr Elmasry*

Computer Science Department, Alexandria University, Alexandria, Egypt

Received 23 January 2003; received in revised form 19 December 2003; accepted 19 January 2004
Communicated by P. Spirakis

Abstract

We give a new, simple proof for the sequential access theorem for splay trees. For an n-node splay tree, our bound on the number of rotations is $4.5n$, with a smaller constant than the bound of $10.8n$ concluded by Tarjan. We extend our proof to prove the deque conjecture for output-restricted deques. Our proofs provide additional insights into the workings of splay trees.

Keywords: Algorithms; Data structures; Splay trees; Amortized analysis; Combinatorial problems

1. Introduction

A binary search tree is a binary tree whose nodes contain items in symmetric order. In other words, for any node x, all the items in the left sub-tree of x are less or equal to the item in x, and all the items in the right sub-tree of x are greater. The nodes of an n-node tree are identified by their symmetric order numbers, from 1 to n.

The splay tree, introduced by Sleator and Tarjan [1], is a self-adjusting binary search tree, which supports a restructuring operation of the tree called *splay*. The splay operation is implemented as a sequence of rotations of edges. A rotation of an edge maintains the symmetric property. See Fig. 1.

When any node of the tree is accessed, a splay is performed at this node. Let $p(x)$ be the parent of x. A splay at x repeats the following step until x becomes the root of the tree.

* Corresponding author. Department of Computer Science, Rutgers University, New Brunswick, NJ 08903, USA.
E-mail address: elmasry@cs.rutgers.edu (A. Elmasry).

0304-3975/$ - see front matter
doi:10.1016/j.tcs.2004.01.019

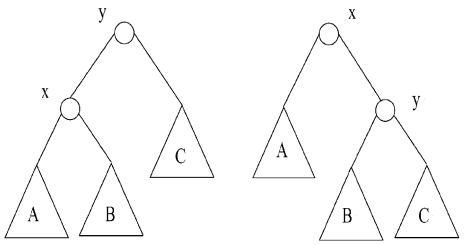

Fig. 1. Rotation of the edge joining x and y.

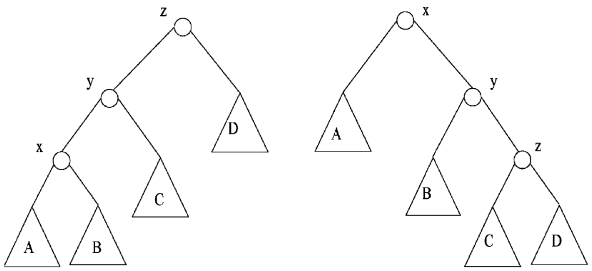

Fig. 2. The zig–zig case.

Zig case: If $p(x)$ is the root: Make x the new root by rotating the edge joining x and $p(x)$.

Zig–zig case: If x and $p(x)$ are both left or right children: Rotate the edge joining $p(x)$ to its parent, and then rotate the edge joining x to $p(x)$. Because of its special interest in this paper, we show this case in Fig. 2.

Zig–zag case: If x is a left child and $p(x)$ is a right child or vice versa: Rotate the edge joining x to $p(x)$, and then the edge joining x to its new parent.

It is easy to use splaying to implement such search tree update operations as insertion and deletion as well as joining and splitting [1]. Sleator and Tarjan [1] also proved that, in an amortized sense, the splay tree is as efficient as any balanced search tree and the static optimal binary search tree, when used for processing a sequence of dictionary operations. They made an even stronger conjecture known as the *dynamic optimality conjecture*. Consider any sequence of accesses, and suppose we carry out the accesses by beginning with an arbitrary binary search tree and searching it from the root for the desired items in the desired order, with the provision that between accesses we can change the tree by performing arbitrary rotations. The total cost of the access sequence is the total number of nodes on access paths plus the total number of rotations. Let $T(s)$ be the minimum total cost of access sequence s for any such binary search tree algorithm.

Dynamic optimality conjecture. *If s is any access sequence, then the cost of performing s by using splaying is $O(T(s)+n)$, for any n-node initial tree.*

If this conjecture is true, splay trees are a form of universally efficient search trees. The sequential access theorem for splay trees is a special case of the above conjecture.

Sequential access theorem. *If we access each of the n nodes of an arbitrary initial tree once, in symmetric order, the total time spent is* $O(n)$.

Tarjan [3] proved the sequential access theorem, bounding the number of rotations with $10.8n$. Sundar [2] gave an easier proof that uses a potential function technique. His bound for the number of rotations is $15n$. In this paper, we give a new proof for the sequential access theorem, bounding the number of rotations with at most $4.5n$.

A deque (double-ended queue) is an abstract data structure, on which the following operations can be performed:

- *push*(e): add item e to the front of the queue,
- *pop*: remove the front item from the deque and return it,
- *inject*(e): add item e to the rear of the deque,
- *eject*: remove the rear item from the deque and return it.

The deque conjecture (Tarjan [3]). *If we perform a sequence of m deque operations on an arbitrary n-node splay tree, the total time is* $O(n+m)$.

When the dynamic optimality conjecture is extended to include update operations it implies the deque conjecture. Sundar [2] gave an inverse Ackermann upper bound of $O((n+m)\alpha(n+m))$ for the deque operations. Tarjan [3] proved a special case of this conjecture for output-restricted deques, i.e. no *eject* operations are allowed. He gave a bound of $11.8n + 14.8m$ rotations. Both Tarjan and Sundar suggested implementing the deque operations as follows. To carry out *push*(e), the current tree becomes the right sub-tree of item e, which becomes the new root. To carry out *pop*, left child pointers are followed until reaching a node x with no left child, a splay operation is then performed at node x removing it from the tree. The implementations of *inject* and *eject* are symmetric.

Extending our proof of the sequential access theorem, we easily prove the deque conjecture for the output-restricted deques, improving over Tarjan's constant. Our bound on the number of rotations is $4.5n + m$. More interesting is to try to extend our proof to prove the deque conjecture in its general form. The paper is interesting in two aspects. Firstly, because it improves the constants, and secondly, because it follows an interesting proving procedure, using a coloring framework, which is surprisingly simple and efficient. Furthermore, the proof gives more insights on how the splay operation is efficient as a restructuring heuristic, demonstrating the strength of splaying.

2. The proof

In a binary tree, the left spine of a sub-tree is defined to be the path from the root of this sub-tree to its leftmost leaf. In other words, every node on the path is the left child of its predecessor. The right spine is defined analogously.

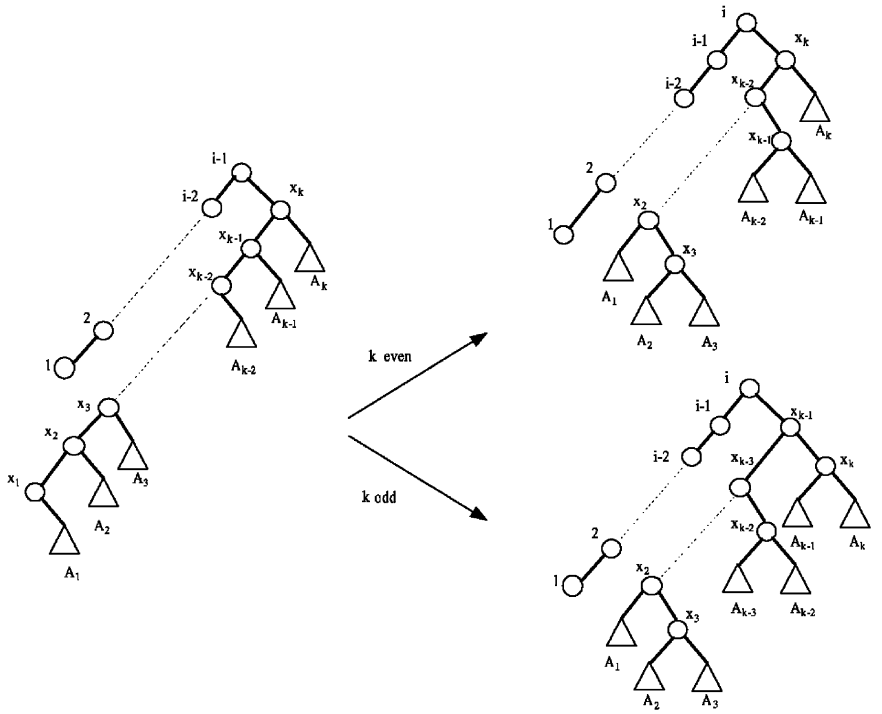

Fig. 3. The effect of the ith splaying. The splaying path contains nodes $x_1, x_2, \ldots, x_k$.

We may think about the sequential access theorem for splay trees as repeated splaying on the leftmost leaf of the right sub-tree of the root. (The first splay operation is an exception being performed on the leftmost leaf of the tree.) As a result of a splaying operation on such node, this node becomes the root of the tree and the old root becomes the root of the left sub-tree. Hence, we may ignore the left sub-tree (which is always a path) entirely and only keep track of the right sub-tree. We call the left spine of the right sub-tree of the root, the *splaying spine*.

Before a splay operation, name the nodes on the splaying spine x_i, such that x_i is the left child of x_{i+1}, for all i starting from 1. As a result of a splaying operation performed on x_1, the following restructuring takes place: The node x_1 becomes the root of the tree, leaving, on the splaying spine, its right child and the nodes on the left spine of this right child's sub-tree. For every odd value of $i \geqslant 3$, x_i is linked to x_{i-1} as its right child, and the right sub-tree of x_{i-1} becomes the left sub-tree of x_i. See Fig. 3.

For the purpose of the proof, we use a coloring scheme to distinguish some nodes from others. The following coloring rules are applied:

- initially, all the nodes are uncolored,
- when an uncolored node becomes a node on the splaying spine, it is colored yellow,
- when a yellow node is linked to another node, this yellow node is colored green,
- when a green node is linked to a yellow node, the yellow node is colored green,

- all the nodes on the right spine of the tree are colored black, overriding the above coloring rules.

As a result of these coloring rules, we can deduce the following:

- the nodes on the splaying spine are colored either yellow or green,
- except for the black nodes that lie on the right spine of the tree, any node is first colored yellow then green. Once colored green it remains green,
- the parent of a colored node is a colored node as well,
- at the moment when nodes are colored yellow, they must be the deepest (the bottommost) nodes of the splaying spine.

The splay operations are numbered, starting from 1. We assume that the splay operation number t takes place at time t. Consider any node x in the tree. Let $g_x(t)$ be the number of the colored nodes on the right spine of x, after the splay operation t. Let $h_x(t)$ be the number of the colored nodes on the right spine of the left child of x, after the same splay operation. If x does not have a left child, then $h_x(t)$ is equal to 0. Before any splay operation, $g_x(0) = h_x(0) = 0$. Define $v_x(t)$ to be equal to $g_x(t) - h_x(t)$.

Consider any node w and its left child z on the splaying spine at time t, such that w is linked to z during the splay operation $t+1$, for any $t \geqslant 0$. The following relations hold:

$$h_w(t) = g_z(t), \tag{1}$$

$$g_w(t+1) = g_w(t), \tag{2}$$

$$h_w(t+1) = h_w(t) - 1, \tag{3}$$

$$g_z(t+1) = g_w(t) + 1, \tag{4}$$

$$h_z(t+1) \leqslant h_z(t) + 1. \tag{5}$$

The equality in relation (5) always holds, except for the special case when z is x_2 (i.e. the left child of z is the node we are splaying at, which has no left children). For such a special case, we have $h_z(t+1) = h_z(t) - 1$.

The following lemma relates the color of the nodes to their v values.

Lemma 1. (a) *For any yellow node* x, $v_x \geqslant 0$.
(b) *If* x *is green, then* $v_x > 0$.
(c) *If a green node* w *is linked to a node* z *at time* $t+1$, *then* $v_z(t+1) > v_z(t)$.

Proof. We prove the lemma by induction on the lifetime of nodes. When a node x is colored yellow at time t_0, the base case follows from the fact that $g_x(t_0) = 1$ and $h_x(t_0) = 1$ (except for the deepest node of the splaying spine that will have $h_x(t_0) = 0$). If x is nonblack and is not involved in a rotation, then v_x will not change. Consider a node w that is linked to a node z at time $t+1$. If w is black at time t (w is on the right spine of the tree), z becomes black at time $t+1$, and the conditions of the lemma do not apply to w and z anymore. Hence, we may assume that both w and z are either yellow or green. Using the induction hypothesis for w at time t, then $v_w(t) \geqslant 0$.

Subtracting (3) from (2), then $v_w(t+1) > v_w(t)$, which implies $v_w(t+1) > 0$. Since w is green after this link, the hypothesis is true for w at time $t+1$. Adding (1) and (4) implies

$$g_z(t+1) = v_w(t) + g_z(t) + 1. \tag{6}$$

Two cases follow depending on the color of w at time t:

Case 1: w is yellow at time t: Using the induction hypothesis for w at time t, $v_w(t) \geqslant 0$. It follows from (6) that $g_z(t+1) \geqslant g_z(t) + 1$. Subtracting (5) from the latter relation, then $v_z(t+1) \geqslant v_z(t)$, and v_z is nondecreasing with time. Since the color of z does not change as a result of such a link, the hypothesis is true for z at time $t+1$.

Case 2: w is green at time t: Using the induction hypothesis for w at time t, $v_w(t) > 0$. It follows from (6) that $g_z(t+1) > g_z(t) + 1$. Subtracting (5) from the latter relation, then $v_z(t+1) > v_z(t)$, proving part (c) of the lemma. Using the induction hypothesis for z at time t, $v_z(t) \geqslant 0$. It follows that $v_z(t+1) > 0$. Since z is green after this link, the hypothesis is true for z at time $t+1$. □

Other than the links that involve black nodes, there are four possible types of links: yellow-to-yellow, yellow-to-green, green-to-yellow and green-to-green. For the first three types, the color of one yellow node changes to green. Once colored green it remains green. This bounds the count of these three types of links with at most n. What is left is bounding the green-to-green links. Using part (b) of Lemma 1, for any green node z, $v_z > 0$. We distinguish between two types of green-to-green links. If a green node is linked to z while $v_z = 1$, we call this link a green-to-green A-link. Otherwise, if a green node is linked to z while $v_z \geqslant 2$, we call this link a green-to-green B-link. Using part (c) of Lemma 1, when a green node is linked to a node z, the value of v_z increases. Hence, any green node z may gain at most one child by a green-to-green A-link before v_z becomes at least 2, bounding the count of the green-to-green A-links with at most n.

Bounding the green-to-green B-links is more involved. We use the accounting method [4] for bounding the number of such links. In such a method we allocate credits in the data structure. These credits are used later on to pay for the structure's operations, one credit per operation. If the allocated credits are enough to pay for all the operations, the number of such credits is used as an upper bound on the cost of the corresponding operations. Consider a green node w that is linked to a green node z, whose $v_z(t) \geqslant 2$, at time $t+1$. The problem is that both w and z may be involved in several links after this one, while their v values are at lease 2, as a result of both of them keep coming back to the splaying spine. The question is how many credits should we allocate per node to cover such links. What we are looking for is to allocate a number of credits on such nodes that would represent some structural behavior of the splay tree. When the structure changes as a result of such a link, the required number of credits should decrease. More precisely, for the case of such a green-to-green B-link, the difference between the credits allocated to w and z before the link and the required number of credits on w and z after the link should be at least 1. This extra credit is then used to pay for that link, while the remaining credits are enough to pay for the succeeding operations. We keep the invariant that, after the splay operation t, there are $h_x^2(t)/2$

credits on any node x. Next, we show that these credits are enough to pay for all the green-to-green B-links, while maintaining the invariant. Let d be the difference between the sum of the number of credits on w and z before the splay operation $t+1$ and those needed after the splay operation $t+1$. It follows that

$$d = \frac{h_z^2(t)}{2} + \frac{h_w^2(t)}{2} - \frac{h_z^2(t+1)}{2} - \frac{h_w^2(t+1)}{2}.$$

Using (3) and (5), then $d \geqslant h_w(t) - h_z(t) - 1$. Using (1), then

$$d \geqslant v_z(t) - 1.$$

Since $v_z(t) \geqslant 2$ for the green-to-green B-links, it follows that for such links $d \geqslant 1$. This extra credit is used to pay for such a link. We still need to maintain these credits for other types of links. If $v_z(t) = 1$, then $d \geqslant 0$ and the credits are reserved. Alternatively, if $v_z(t) = 0$ (z is a yellow node), then $d \geqslant -1$. In this case, to maintain the invariant for the number of credits, one credit is needed. Since, for this type of links, the color of a yellow node becomes green, the shortage for this credit is paid-for by such a yellow node. It follows that we need to allocate one credit per node at the time when this node is firstly colored yellow. Additionally, when a node x is colored yellow, the value of h_x is 1 and $h_x^2/2$ equals $\frac{1}{2}$. Hence, to maintain the invariant for each of these nodes, another half credit is allocated to each node when it is firstly colored yellow, for a total of 1.5 credits per yellow node. The total number of the allocated credits is, therefore, $1.5n$, and the count of the green-to-green B-links is bounded by $1.5n$.

When the number of nodes on the splaying spine is even, the root of the splaying spine is not linked to another node. The relations defined earlier may not hold for this node. Fortunately, this node is colored black. The black nodes are involved in at most one link per splay operation (A black node is involved in a link when the number of nodes on the splaying spine is odd.). This costs at most n extra links.

The $4.5n$ bound on the number of rotations for the sequential access theorem follows by adding the following bounds: The n links that involve at least one yellow node, the n links bounding the green-to-green A-links, the $1.5n$ links bounding the green-to-green B-links, and the n links that involve black nodes. The following theorem follows.

Theorem 2. *The total number of rotations involved in sequentially accessing the n nodes of an initial arbitrary splay tree is at most* $4.5n$.

Now, consider the implementation of the output-restricted deque operations using splay trees. For making the proof easier we change the implementations of Tarjan and Sundar for the *inject*(e) operation. For the new implementation, right pointers of nodes starting with the root are followed, until a node that has no right child is reached. The node e is then appended as this node's right child. (For ease of implementation, a pointer may be kept pointing to the rightmost leaf in the tree.) When a new node is pushed or injected, it lies on the right spine of the tree, and hence it is colored black. Every black node accounts for only one rotation, for a total of at most m extra rotations. The proofs of the above lemmas still hold, and the following lemma follows:

Lemma 3. *Consider a sequence of m output-restricted deque operations performed on an arbitrary n-node splay tree. The deque operations require a total of at most $4.5n + m$ rotations.*

Note that, if the *inject* operation is implemented as in [2,3], the right spine of the tree changes, and Lemma 1 holds for neither the injected node (current root of the tree) nor the nodes on the right spine of its left child (previously black nodes). In this case, we can apply Tarjan's proof [3], while using our bound on the number of rotations for the sequential access theorem. Briefly, the basic idea of the proof in [3] is to divide the process into epochs, every epoch ends when the original tree of that epoch as well as the pushed nodes are popped. The newly injected nodes remain at the end of the epoch, representing the new tree of the next epoch. Using our bound of $4.5n$ rotations for the sequential access theorem, this leads to a bound of $6.5n + 9.5m$ on the number of rotations for the output-restricted deque operations. See [3] for the details.

References

[1] D. Sleator, R. Tarjan, Self-adjusting binary search trees, J. Assoc. Comput. Mach. 32 (3) (1985) 652–686.
[2] R. Sundar, On the deque conjecture for the splay algorithm, Combinatorica 12 (1992) 95–124.
[3] R. Tarjan, Sequential access in splay trees takes linear time, Combinatorica 5 (1985) 367–378.
[4] R. Tarjan, Amortized computational complexity, SIAM J. Algebraic Discrete Methods 6 (1985) 306–318.