A simple bound for sequential access in splay trees

Colm Ó Dúnlaing*
Mathematics, Trinity College, Dublin 2, Ireland

(Revised 2010)

Abstract

This note improves upon an earlier analysis of the author's [2]. Using methods from that paper, we give a different proof that the cost of traversing a binary tree by repeated splay-to-root is $O(n)$, a result originally due to Tarjan [5].

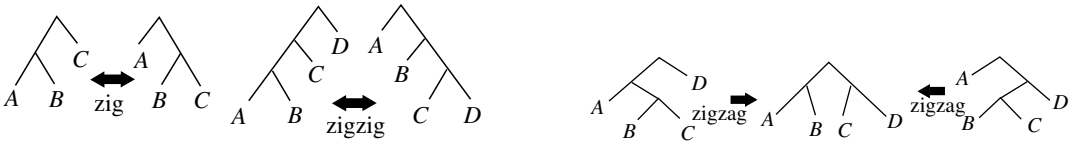
1 Introduction

Splay trees are binary trees continually modified by 'splaying' [4]. It was established by Tarjan in 1985 [5] that the cost of traversing by repeated splay-to-root is $O(n)$. Since then there have been several other proofs of varying strength and difficulty: it is now known that the cost is at most $4.5n$ [1]. Related problems are still under investigation [3].

The present author derived a weaker estimate through a detailed analysis of various recurrences [2]. This note extends that analysis to derive an $O(n)$ estimate using a method different from those already published.

2 Definitions

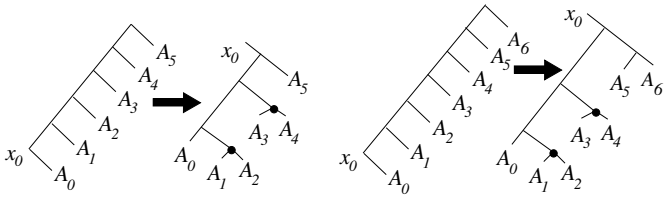
To *splay* a node, not the root, in a binary tree, means to bring it closer to the root by one rotation if its parent is root, else two, as illustrated: see [4].



We consider a binary tree being traversed by *fetching* the nodes in inorder, by which we mean repeatedly bringing the nodes to the root by splaying.

$T(s)$ denotes the tree after s fetches. The *spine*, spine(s), is the maximal left branch in $T(s)$ containing the next node, the $(s+1)$st, to be fetched. Analysis reveals that each fetch modifies the tree as illustrated (depending on the parity of |spine(s)|).

*e-mail: odunlain@maths.tcd.ie. Mathematics department website: http://www.maths.tcd.ie.



We use the following notation: $x_0, \ldots, x_r$ are the spine nodes, and x_0 will be fetched next; A_i is the right subtree of x_i. First, fetching: splaying causes x_0 to be fetched, becoming the root. Second, halving: for $i \geq 1$, x_i is *displaced* from the spine if i is even. That is, the arrangement $x_{i-1}A_{i-1}x_iA_i$ is adjusted so that x_{i-1} has right subtree $A_{i-1}x_iA_i$. Third, extending: A_0 becomes the left subtree of x_1 (extending the spine if A_0 is nonempty). Because of halving,

$$|\text{spine}(k) \cap \text{spine}(k+1)| = \lfloor |\text{spine}(k)|/2 \rfloor. \tag{2.1}$$

It can be shown (e.g., [2]) that the highest nodes on the spine have been there longest; at the bottom is a sequence of 'fresh' nodes which have just been introduced; and in-between are 'recycled' nodes which have been on the spine before, pushed off it, and restored to the spine. (Any of these sets could be empty.)

The overall traversal cost (in rotations) is

$$\left(\sum_{s=0}^{n-1} |\text{spine}(s)| \right) - 1. \tag{2.2}$$

A (spine) *block* (B, s) [2, Definition 2.3] is a subinterval of consecutive nodes on spine(s). Sometimes the sequence B itself will be called a block, with s understood. For the sake of generality, B can be empty.

For every node u, there is an earliest s such that $u \in \text{spine}(s)$. Let B_s be the block of nodes whose first occurrence is on spine(s), i.e., the 'fresh' nodes on spine(s). If $B_s \neq \emptyset$ then B_s is leftmost on the spine (i.e., contains the leftmost node in spine(s)).

(2.3) Definition *C_s is the set of nodes in* spine(s) *but not in* spine$(s-1)$ *(if $s > 0$). That is, $C_0 = B_0 =$* spine(0) *and if $s > 0$ then C_s contains all nodes brought onto the spine by 'extending' as described above. C_s is a block on* spine(s) *and it contains B_s (leftmost).*

(2.4) Lemma *Let $c_s = |C_s|$. The total spine length, and therefore the traversal cost, is $\leq 2 \sum c_s$.*

Proof.

$$|\text{spine}(0)| = c_0$$

$$|\text{spine}(s+1)| \leq \frac{|\text{spine}(s)|}{2} + c_{s+1} :$$

$$\sum_s |\text{spine}(s)| \leq (c_0 + c_0/2 + c_0/4) + (c_1 + c_1/2 + c_1/4) + \ldots + c_n$$

$$< 2(c_0 + c_1 + \ldots + c_n). \quad \blacksquare$$

Repeated splay-to-root operations compress blocks into shallow binary trees. As defined in [2, Section 3], given a node v in a block (B, s), it goes through several *generations*: When v first joins the spine, until it first leaves the spine, it is first generation — relative to (B, s); if not fetched it becomes second-generation, remaining so while it rejoins the spine and until it next leaves the spine, and so on, until it is fetched.

Given any maximal left branch in some $T(t)$, those nodes in the branch which belong to B form an *offline branch*, provided the intersection is nonempty; an offline *child fragment* is an offline branch which contains at least one second-generation node[1]: and when an offline child fragment gets pulled back onto the spine it becomes an (online) *child fragment*. Recursively this spawns *descendant fragments* (with parents and ancestors in the obvious way). The 'descendant' relation is a reflexive.

There is an obvious thinning-out effect which dictates that a fragment is usually much shorter than its parent, but not always so.

(2.5) Definition *A fragment F is* short *if $|F| \leq 2$,* long *if $|F| \geq 3$.*

Suppose that (F, s) is a child fragment of (G, r). F is exceptional *if $F = G$, i.e., the nodes in G recur in* spine(s).

(2.6) Proposition (i) *Only short fragments can be exceptional.* (ii) *There exists a constant γ such that for every block B, the total length of all its non-exceptional descendant fragments is $\leq \gamma|B|$* [2, Corollary 6.5]. ∎

The qualifier 'non-exceptional' is needed because a short fragment can recur arbitrarily often.

(2.7) Corollary *The total length of all long descendant fragments, together with their short children, is at most $3\gamma|B|$.*

Proof. If F is a long fragment, then it has at most $|F|$ first-generation children relative to F. Counting only the short ones, the total length is $\leq 2|F|$. Summing over all long descendants F, it adds at most $2\gamma|B|$ to the total length. ∎

Colour scheme. In Lemma 2.4 C_s is the set of nodes x where x is introduced or restored to the spine at time s. Note $C_s \supseteq B_s$ and C_s occurs leftmost in spine(s). Let us colour the nodes $x \in C_s$ green if $x \in B_s$ or the fragment (descendant of some earlier block) containing x is a long fragment, otherwise red. Nodes in spine$(s) \cap$ spine$(s-1)$ are colourless.

By the above proposition the total number of green nodes (summed over all s) is at most γn.

It remains to count the red nodes. At each time-step s, the spine contains an alternating set of red and green blocks. Each block will be allocated a set of credits which so that the total credit bounds the total size of all red blocks.

The 'child fragment' relation prompts us to think of the descendant fragments of a (nonempty) block B as a tree. From Proposition 2.6, the long fragments form a prefix subtree whose total fragment length is $\leq \gamma|B|$.

[1] Offline nodes are second generation and higher.

The credit system below, allotted to red and green blocks, is intended to cover the cost (i.e., length) of all red blocks.

The coloured nodes in a spine occur alternating in red and green intervals. Let us call each such interval a red or green *band*. A band is a block under a new name. The red bands will be organised into *clusters*. Clusters are trees whose nodes are red bands. Each cluster has a root band, and a child of a band is a fragment in the earlier sense. Now, however, fragments may be absorbed in new clusters.

All the credits will first be allocated to the fetch steps and the green bands, subsequently transferred to the red bands. This transfer is legitimised by the following easy

(2.8) Lemma *If (X, s) and (Y, s) are blocks and $X \subseteq Y$, then every fragment of X is contained in a fragment of Y.* ∎

The allocation scheme is dictated by Corollary 2.7. From a red band R one expects at most $3\gamma|R|$ red nodes in long descendant fragments together with any short children. Thus a root band R must receive $3\gamma|R|$ credits.

A red band R is designated a root band if it is long, and it is a concatenation of fragments $F_1F_2 \dots F_r$, where either $r = 1$ and the parent of F_1 is green, or $r > 1$.

So, when creating a new cluster rooted at R, for each fragment F_i it is required to obtain $3\gamma|F_i|$ credits from elsewhere. There are several cases.

- Case 1. The parent G_i of F_i is green. Take $3\gamma|F_i|$ credits from G_i.
- Case 2. G_i is long and red. Take $3\gamma|F_i|$ credits from the root of the cluster containing G_i (F_i and its descendants are deleted from that cluster).
- Case 3. G_i is short (and red). Take $3\gamma|F_i|$ credits from G_i. Note $|G_i| \leq 2$ and its child fragments add up to at most $|G_i|$.

We need $3\gamma|B| + |B|$ credits for each short red band B, $|B|$ for the band itself and the rest to cover Case 3. If there are k red bands in C_s, then there are at least $k - 1$ green bands. If $2 + 6\gamma$ credits are allocated to the fetch step and to each of the $k - 1$ green bands, they can be transferred to the adjacent short red bands.

Case 1 can be covered with by allocating $3\gamma|B|$ more credits to each green band B. Thus $(2 + 6\gamma)n + (2 + 9\gamma)T$ credits are enough, where T is the total number of green nodes. But $T \leq \gamma n$, so in the notation of Lemma 2.4,

$$\sum c_s \leq \delta n \quad \text{where} \quad \delta = 2 + 8\gamma + 9\gamma^2.$$

There are at most $2\delta n$ rotations (Lemma 2.4). This is a new $O(n)$ estimate, but the constant is huge ([2, Section 6]).

3 References

1. Amr Elmasry (2004). On the sequential access conjecture and deque conjecture for splay trees. *Theoretical Computer Science* **314:3**, 459–466.

2. Colm Ó Dúnlaing (2003). Inorder traversal of splay trees. *Electronic Notes in Theoretical Computer Science* **74**, 24 pages.

3. Seth Pettie (2008). Splay trees, Davenport-Schinzel sequences, and the deque conjecture. *Proc. 19th ACM-SIAM Symposium on Discrete Algorithms,* 1115–1124.

4. Robert E. Tarjan (1983). *Data structures and network algorithms.* SIAM.

5. Robert E. Tarjan (1985). Sequential access in splay trees takes linear time. *Combinatorica* **5:4**, 367–378.