6.851: ADVANCED DATA STRUCTURES, SPRING 2021

Prof. Erik Demaine, Josh Brunner, Dylan Hendrickson, Yevhenii Diomidov

Problem Set 1 Solutions

Due: Thursday, Feburary 25, 2021

Problem 1.1 [Constant-Time Concatenate].

Solution:

The data structure to support constant-size concatenate and $O(\log n)$ BST operations is essentially a 2-3 tree. Our data structure, $D = (B, p_{\text{min}}, p_{\text{max}})$ will be a 2-3 tree B, with backpointers and two specific pointers $p_{\text{min}}, p_{\text{max}}$ to the min/max nodes. All of the BST operations are performed the same way as in a 2-3 tree while maintaining backpointers and updating $p_{\text{min}}, p_{\text{max}}$ when necessary. This takes constant overhead. For concatenate, one does the following for height $h(B) \leq h(B')$:

concatenate$((B, p_{\text{min}}, p_{\text{max}}), (B', p'_{\text{min}}, p'_{\text{max}}))$:

(a) Delete p_{max} of B in amortized constant time using the pointer.

(b) With p'_{min}. Travel up backpointers to the height $h(B) + 1$ ancestor of p'_{min}.

(c) Merge p_{max} to the left of the $h(B) + 1$ node and let B be it's leftmost tree.

(d) Rebalance if necessary.

The case $h(B) > h(B')$ is done symmetrically. The cost of a concatenate is $h(B)$+rebalance cost. This motivates the following potential function for our data structure to prove amortized constant time:

$$\Phi(D) = h(B) + \#\text{nodes that are full}.$$

Doing a potential function analysis, we first find that all of the update/balancing operations for a 2-3 tree still happen in amortized constant time with respect to this potential function. The typical potential function used in such arguments is the number of full nodes and the height of a 2-3 tree can change by at most 1 after balancing. Thus, we can conclude that **concatenate** takes amortized constant time from the following calculation:

$$\begin{aligned} T_{\text{amort}} &= 1 + h(B) + \text{rebalance} + \Delta\Phi && (1) \\ &= [1 + h(B) + h(B') \pm 1 - (h(B) + h(B'))] + [\text{rebalance} + \Delta\#\text{nodes that are full}] && (2) \\ &= O(1). && (3) \end{aligned}$$

Since 2-3 trees are always balanced, search still takes $O(\log n)$ and thus predecessor, successor, and the searches for delete/insert all take $O(\log n)$ time.