6.851 Advanced Data Structures (Spring'10)

Prof. Erik Demaine Dr. André Schulz TA: Aleksandar Zlateski

$\boxed{\text{Problem 6}}$ *Due: Thursday, Mar. 18*

Be sure to read the instructions on the assignments section of the class web page.

Dynamizing static search structures. An *exponential search tree* on n keys is a search tree data structure where the root has $\Theta(n^{1/c})$ children, for a constant $c > 1$, and each child subtree is an exponential search tree on $\Theta(n^{1-1/c})$ keys. The actual data is stored at the leaves. Internal nodes store "splitters", which are or were keys, to enable search. Namely, an internal node stores a splitter for each child that is less than or equal to all nodes in the child subtree, and greater than all nodes in its left sibling subtree. For example, the splitter for a child could be the smallest key in the child subtree. (Assume for simplicity that all keys are distinct.) The internal node stores these splitters in a specified static predecessor search data structure (such as fusion trees).

(a) Describe how to perform a predecessor or successor search in an exponential search tree on n keys, and derive a recurrence for the time $T(n)$ in terms of the time $S(k)$ to search in a static structure on k keys. Show that for fusion trees, where $S(k) = O(\log_w k)$, $T(n) = O(S(n))$.

(b) Prove that the space $C(n)$ required by an exponential search tree on n keys is $O(n)$.

(c) Describe how to perform an insert or delete in an exponential search tree on n keys, and prove that the amortized cost is the cost $T(n)$ of an initial search plus $O(\lg\lg n)$. You should maintain a property called *strong weight balance*: the total number of nodes in a subtree rooted at a node of depth d (where the root has depth 0) should be $\Theta(n^{(1-1/c)^d})$ at all times.[1] Assume that you can build a static structure on k keys in $B(k) = k^b$ time, for a given constant $b \geq 1$. You can choose c accordingly.

Congratulations, you've dynamized fusion trees!

[1]Hint: mimic B-trees where, when a node becomes a constant factor larger or smaller than its mean desired size, you split the node or merge it with an adjacent sibling. Merging with a sibling may cause an immediate split because the joined node is too full, but this split will not cause another merge if the overflow/underflow thresholds don't overlap.