CS 224 Advanced Algorithms — Fall 2014

Problem Set 3

Due: 11:59pm, Monday, October 6th
Submit to: `cs224-f14-assignments@seas.harvard.edu`
Solution maximum page limit: 5 pages

See homework policy at `http://people.seas.harvard.edu/~minilek/cs224/hmwk.html`

Problem 1: Consider splay trees. For any access sequence $\sigma = (x_1, x_2, \ldots, x_m)$ for each $i \in \{1, \ldots, n\}$ and fixed binary search tree T, let $C_T(\sigma)$ denote the cost of servicing σ with T. Let $S(\sigma)$ be the cost of servicing σ with a splay tree. We showed in class that $S(\sigma) = O(m + n^2 + C_T(\sigma))$.

(a) (7 points) Modify the weight function we used in class to show that in fact $S(\sigma) = O(m + n \log n + C_T(\sigma))$. As in the analysis in class, your proof should not use the fact that the optimal tree achieves the entropy bound.

(a) (3 points) Deduce that if each $i \in \{1, \ldots, n\}$ appears in σ at least once, then $S(\sigma) = O(m + C_T(\sigma))$.

Problem 2: (5 points) Prove Lemma 2 from the Lecture 6 notes, namely that a tree of rank k in a Fibonacci heap has at least F_{k+2} nodes, where F_k is the kth Fibonacci number.

Problem 3: (10 points) In Fibonacci heaps, when a node x loses 2 children, the subtree rooted at x is cut from x's parent and becomes a new tree in our top level forest. Suppose that instead we cut x's subtree away from its parent only after x loses k children.

(a) (5 points) Show that the amortized cost of decrease key is reduced as k increases. How does it decrease as a function of k? Note decrease key already has amortized cost $O(1)$ when $k = 2$, so the point here is just that the constant inside the big-Oh improves. **Hint:** modify the potential function from class.

(b) (5 points) Which operation(s) increase in amortized cost due to this change? Give a new bound as a function of k.

Problem 4: We will explore in this problem another approach to obtain $O(\log n)$ amortized bounds for insertion, deletion, and find in a binary search tree (BST).

1. (3 points) Suppose we start with $n = 2^k - 1$ items in a perfect balanced binary search tree and we only wish to support deletion and find. We augment the tree so that each node x stores a mark bit $\textsc{mark}(x)$, initialized to 0, as well as $\textsc{size}(x)$, the number of nodes in x's subtree (including x). When we are asked to delete a node, we set its mark bit to 1 without actually deleting it then decrement the size field of that node and all

its ancestors by 1. After 2^{k-1} deletions, we rebuild a brand new perfectly balanced binary search tree on the set of unmarked nodes. Describe how to implement find in this data structure and show that the rebuilding procedure can be implemented so that the amortized cost of deletion is $O(\log n)$.

2. (7 points) Suppose we only want to support insertion and find. We pick some fixed constant $\alpha > 1$, and again BST nodes are augmented to store sizes of subtree. Furthermore each node is augmented to know its "height", i.e. the absolute difference between its own depth in the tree and the depth of its deepest descendant node. We insert a key x as in a normal BST. Then once x is placed in some leaf, if any ancestor node v of that leaf is such that $\textsc{height}(v) > \alpha \cdot \lceil \log_2(\textsc{size}(v)) \rceil$, then we rebuild the entire subtree rooted at v to make it as perfectly balanced as possible. Show that this scheme can be implemented so that find takes $O(\log n)$ worst case time, and insertion takes amortized $O(\log n)$ time. How do the constants in the big-Ohs behave as a function of α?
Hint: define an appropriate potential function for each node, and the potential of the tree as the sum of potentials of nodes.

Note that the above two schemes can be combined to support all three operations (insertion, deletion, and find) in $O(\log n)$ time (the running time for find is worst case, whereas the other two operations have amortized bounds).

Problem 5: (1 point) How much time did you spend on this problem set? If you can remember the breakdown, please report this per problem. (sum of time spent solving problem and typing up your solution)