Algorithms
Computer Science 140 & Mathematics 168
Spring 2007
Homework 6a
Due Thursday, February 22

- Exam 1 will be given out in class on Thursday, March 1 and will be due on Tuesday, March 6 at 5 PM. The exam has no time limit. You may use class-notes that you have either taken in class and/or prepared before the exam (hand-written or typeset by you). No other materials or sources may be used.

1. **[10 Points] Deleting Elements from 2-3-4 Trees.**

   In class we described algorithms for both finding and inserting an element into a 2-3-4 tree. In this problem, we will take a look at a deletion algorithm and you will then try it out!

   First, take a moment to review the algorithm for deleting an element x in a standard binary search tree. In particular, recall that if x is a leaf or has just one child, the situation is easy. If x has two children, we first find its successor y, remove y, and replace x with y.

   Now, let's consider removing an element x from a 2-3-4 tree. If x is in a leaf node, we can remove it and then do a little trickery to make sure that the tree remains a legitimate 2-3-4 tree. (More on this in a moment.) If x is not in a leaf, we can find its successor y and replace x with y. Notice that y will always be in a leaf. (Make sure you see why.) So now we must remove y. That is, we have reduced the problem of removing a key from a non-leaf node to that of removing a key from a leaf node.

   After removing a key from a leaf (either x if it was in a leaf, or y if x was not in a leaf and we thus found its successor), we must quickly correct the tree and make sure it is a 2-3-4 tree.

   Here is an algorithm to delete a key x. Note that we distinguish between a *key* and a *node*. A node in a 2-3-4 tree may hold 1, 2, or 3 keys.

   - If x is not in a leaf node, find the successor y of x, replace x by y, and remove y from the tree rather than x.
   - Let z be the key to be removed (either x or its successor y, but a key in a leaf in any case). If z has other keys in its node, then z can simply be removed.

- If z is the only key in its node but has an adjacent sibling node (a node may have multiple siblings but an adjacent sibling is one immediately to its right or left) with more than 1 key, identify the key u (in an adjacent sibling) that is numerically closest to z, move u up to its parent, and move the appropriate key from the parent down to the leaf to replace z.
- If z is the only key in its node but each adjacent sibling node of z contains only one key, we have an *underflow* - a situation in which the removal of z leaves us with a node with no keys. We fix this by merging or "fusing" this node with one of its adjacent sibling nodes and an appropriately selected key from the parent. This may either take care of the problem or it may create an underflow at the parent since now the parent may have no keys remaining. An underflow at the parent is handled by recursively applying the same procedure to the parent.

(a) Starting with an empty 2-3-4 tree, show the tree that results after the numbers 1, 2, 3, 4, 5, 6, 7, 8, and 9 have been inserted in that order. (Careful here, recall from class that we insert by splitting full nodes on the way down!)

(b) Now show the tree after 5 is deleted.

(c) Next, delete 1 and show the resulting tree.

(d) Next delete 6 and show the resulting tree.

(e) Next delete 9 and show the resulting tree.

(f) Finally, explain in a few sentences why the deletion algorithm takes $O(\log n)$ time to delete an item for a tree with n keys.

2. **[10 Points] Red-Black Sorting!** Professor R. Borreal has proposed the following comparison-based sorting algorithm to sort an array of n elements: Starting with an empty red-black tree, insert the elements in the array one-by-one into the red-black tree. Then, perform an inorder traversal of the tree and "dump" these elements into the array - resulting in a sorted array.

   What is the running time of BorrealSort? Explain your analysis briefly. How does it compare to the known lower-bound on comparison-based sorting?