TEL AVIV UNIVERSITY
Department of Computer Science
0368.4281 – Advanced topics in data structures
Spring Semester, 2003/2004

Homework 2, March 29, 2004

Due on April 19 2004. Please keep a copy of your homework and work alone.

1. We perform a sequence of m operations on an (initially empty) heterogenous 2-4 finger search trees (i.e. a finger search tree without level links). Out of these m operations k are concatenations where the ith concatenation concatenates two trees such that the smaller among them contains n_i elements. The other $m-k$ operations are find, insert, delete, and split. The j-th operation among these operations is performed on an element at distance d_j from the beginning of a list containing n_j elements.

a) Prove that using the implementations of the operations as described in class one can perform the sequence in

$$O(\sum_{i=1}^{k} \log(n_i) + \sum_{j=1}^{m-k} \log(\min\{d_j, n_j - d_j\}))$$

time.

b) Improve the bound in the previous question to

$$O(k + \sum_{j=1}^{m-k} \log(\min\{d_j, n_j - d_j\}))$$

time.

2. A 2-3 tree is defined similarly to 2-4 trees except that internal nodes are restricted to have only 2 or 3 children. We implement insertion by adding the new leaf as a child of the appropriate node x. If node x becomes a 4-node we split it into two 2-nodes and we continue the splittings until we hit a 2-node that becomes a 3-node. We implement deletion by removing the corresponding leaf. If its parent was a 3-node it becomes a 2-node and the operation is complete. Otherwise the parent, say y, becomes a 1-node. If y has a 3-node as a sibling it borrows a child from its sibling. If y does not have a sibling which is a 3-node we fuse it with its sibling and continue in the same manner at the parent of y if it becomes a 1-node.

Show a sequence of m insertions and deletions to an n node tree, where each insertion or deletion requires $\Omega(\log n)$ time even if the point of insertion or the item to be deleted are known.

3. AVL-trees (or height balanced trees) are another popular family of balanced binary trees. A binary search tree is an AVL tree if for each internal node u with children v_1 and v_2

$$|height(v_1) - height(v_2)| \le 1$$

where the height of a node is the longest path from that node to a leaf. We maintain with each node the difference between the height of its left child and the height of its right child.

Suggest implementations of insert and delete for AVL trees that take $O(\log n)$ worst-case time.

Does your implementation support a sequence of m insertions in $O(m)$ time ? If so prove that it does and if not provide a counterexample.

Does your implementation support a sequence of m insertions and deletions in $O(m)$ time ? If so prove that it does and if not provide a counterexample.

4. Consider the Jordan sorting algorithm presented in class. Lets call a pair in the lower tree or the upper tree together with all its children a *family*.
a) Prove the following observation:
Suppose we have already traversed the points $x_1, \ldots, x_i$. Let F be a family so far constructed containing the pair $\{x_{i-1}, x_i\}$. Let $x_j \neq x_{i-1}$ be the point among the points in pairs of F such that x_i and x_j are adjacent (assume that there is such a point), i.e., no point in a pair of F lies between x_i and x_j. Let R be the region bounded by the parts of the curve connecting the pairs in F together with the segments of the x-axis interconnecting them. Let x_k, $k > i$, be the point where the curve next enters R. Then either x_k lies between x_i and x_j or the parent pair in F is $\{-\infty, \infty\}$ and x_i and x_k bracket all other finite points in pairs of F. (Use the fact that the curve is a Jordan curve)

b) Show how to use the observation above to simplify the data structures we used to implement the algorithm.