TEL AVIV UNIVERSITY
Department of Computer Science
0368.4281 – Advanced topics in data structures
Fall Semester, 2005/2006

Homework 2, Nov. 29, 2005

Due on Monday December 12. Please keep a copy of your homework.

1. A double ended queue (deque) is a data structure which supports insertions and deletions of items from both sides. The operations push and pop, insert and delete an item, respectively, from one side of the deque. The operations inject and eject, insert and delete an item, respectively, from the other side of the deque. Describe an implementation of a deque via stacks (where a stack supports only push and pop). Simulate each operation on the deque by a sequence of operations on the stacks. Your implementation should require $O(1)$ amortized time per operation and $O(n)$ space. Prove that indeed your data structure achieves this amortized time bound.

2. 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. (Hint, first show that the amortized cost of rebalancing during insertion/deletion and catenation is constant. Then analyze the cost of a split carefully.)

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.

3. 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.

4. Let P be a triangulated simple polygon.
(a) Describe an algorithm that given P and a vertex x of P, finds the segment on every edge of P which is visible from x. That is, for an edge $e \in P$ the required segment $s \subseteq e$ consists of all points $y \in e$ such that the straight line connecting y and x lies entirely inside P. (It is not hard to prove that this set of points indeed is a segment.)
(b) Describe an algorithm that given P and an edge e of P, finds the segment on every edge of P which is visible from some point of e. That is, for an edge $e' \in P$ the required segment $s \subseteq e'$ consists of all points $y \in e'$ for which there exists a point $x \in e$, such that the straight line connecting y and x lies entirely inside P.

(Hint: For both problems there is a fairly straightforward linear time algorithm based on the shortest path algorithm given in class. There may be other methods as well. Give whatever algorithm you find even if it is not linear and analyse it.)

5. Let $A = \{a_1, a_2, \ldots, a_n\}$ be a set of distinct items totally ordered such that $a_i < a_j$ iff $i < j$. Assume that over a long sequence of m accesses to items in A, a_i is accessed $q(i) \geq 1$ times. Describe an algorithm to find a binary search tree T where each item a_i resides in a leaf of T such that if we serve the accesses using T, each time going from the root to the corresponding leaf, the total time its takes to serve the whole sequence is minimized. Prove
1) that your algorithm indeed constructs a tree that minimizes the total access time. 2) An upper bound on the running time of your algorithm for constructing T. (Any polynomial time algorithm would be fine.)