# Problem Set 3

**Due: Wednesday, September 24, 2003.**

Problem 1 is marked **noncollaborative**. You may not discuss it or related material with anyone else.

Problem 5 is marked **optional**. If you solve it, it will be graded, but you will not receive extra credit for it. It will not affect your grade.

**NONCOLLABORATIVE Problem 1.** Augment the van Emde Boas priority queue to support the following operations on integers in the range $\{0, 1, 2, \ldots, u-1\}$ in $O(\log\log u)$ worst-case time each and $O(u)$ space total:

**Find** $(x, Q)$**:** Report whether the element $x$ is stored in the structure.

**Predecessor** $(x, Q)$**:** Return $x$'s predecessor, the element of largest value less than $x$, or null if $x$ is the minimum element.

**Successor** $(x, Q)$**:** Return $x$'s successor, the element of smallest value greater than $x$, or null if $x$ is the maximum element.

---

**Problem 2.** In class we saw how to merge two sorted words with $k$ entries into one sorted word with $2k$ entries in $O(\lg k)$ time, where $2k \leq w/b$, $w$ is the number of bits in a machine word, and $b$ is the number of bits in an entry. Show how to merge two sorted sequences of $r$ sorted words, each with $k$ entries, into one sorted sequence of $2r$ sorted words, each with $k$ entries, in $O(r \lg k)$ time. You may assume that all $2k$ entries are distinct.

**Problem 3.** Bit tricks. Suppose you are given a word containing $k$ entries of $b$ bits each (where $kb$ bits fit in a word).

**(a)** Show how to exchange the first $k/2$ of the entries with the last $k/2$ of the entries in $O(1)$ time (a constant number of word operations).

**(b)** Show how to reverse the entire sequence of $k$ entries in the word in $O(\lg k)$ time.

**Problem 4.** The *least common ancestor* (sometimes called lowest common ancestor) of nodes $v$ and $w$ in an $n$ node rooted tree $T$ is the node furthest from the root that is an ancestor of both $v$ and $w$.

The following algorithm solves the *offline* problem. That is, given a set of query pairs, it computes all of the answers quickly. It makes use a union-find data structure. You may want to review this data structure (cf. [CLRS], Chapter 21), and in particular the union-by-rank heuristic.

Offline LCA: Associate with each node an extra field "name". Process the nodes of $T$ in postorder. To process a node, consider all of the query pairs it is a member of. For each pair, if the other endpoint has not yet been processed, do nothing. If the other endpoint has been processed do a find on it, and record the "name" of the result as the LCA of this pair. After considering all of the pairs, union the node with its parent, and set the "name" of the set representative to be the parent.

We leave it as an exercise (not to be turned in), that this algorithm is correct, and takes $O((n+m)\alpha(n))$ time. (If you haven't met $\alpha$ before, it is an inverse of Ackerman's function and grows VERY, VERY slowly—even slower than $\log^* n$. It is only 4 on the number of particles in the universe.)

Of course, in some instances we would like to find least common ancestors *online*. That is, we aren't told all of the pairs up front; we get queries one at a time.

**(a)** Show how to use the techniques of persistent data structures to preprocess a tree in $O(n \log n)$ time so as to allow LCA queries to be answered in $O(\log n)$ time. Aim for a simple solution here, even if you solve part (b). **Hint:** path compression is messy for the persistent data structure, and is not necessary to achieve $O(\log n)$ time for union and find operations. Note also that nodes have arbitrary indegree, so path copying won't work.

**OPTIONAL (b)** Improve your solution to take $O(n)$ preprocessing time.

**OPTIONAL Problem 5.** Can a van Emde Boas type data structure be combined with some ideas from Fibonacci heaps to support insert/decrease-key in $O(1)$ time and delete-min in $O(\log \log u)$ time?