Problem Set 1

Due: Wednesday, September 14, 2005.

Collaboration policy: collaboration is *strongly encouraged.* However, remember that

1. You must write up your own solutions, independently.
2. You must record the name of every collaborator
3. You must actually participate in solving all the problems. This is difficult in very large groups, so you should keep your collaboration groups limited to 3 or 4 people in a given week.
4. **No bibles.**

Problem 1. Unlike regular heaps, Fibonacci heaps do not achieve their good performance by keeping the depth of the heap small. Demonstrate this by exhibiting a sequence of Fibonacci heap operations on n items that produce a heap-ordered tree of depth $\Omega(n)$.

Problem 2. Suppose that Fibonacci heaps were modified so that a node was cut only after losing k children. Show that this will improve the amortized cost of decrease key (to a better constant) at the cost of a worse cost for delete-min (by a constant factor).

Problem 3. On tradeoffs in the heap operations.

(a) Let P be a priority queue that performs insert, delete-min, and merge in $O(\log n)$ time, and performs make-heap in $O(n)$ time where n is the size of resulting priority queue. Show that P can be modified to perform insert in $O(1)$ amortized time, without affecting the cost of delete-min or merge (i.e. $O(\log n)$ amortized time). Assume that the priority queue does not support an efficient decrease-key operation.

(b) Using the above technique, show that even binary heaps can be modified to support insert in $O(1)$ amortized time while maintaining an $O(\log n)$ time bound for delete-min. Note that binary heaps do not support merge in $O(\log n)$ time.

The bounds asked above can of course be derived from Fibonacci heaps; the goal is to work with the data structures specified rather than introducing complicated new ones.

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 α 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. Keeping a mark bit around in Fibonacci heaps may be wasteful. Suppose that instead, each time I do a cut, I flip a coin to decide whether to cascade that cut to the parent. If the coin is unbiased, show that the expected behavior of these markless Fibonacci heaps is like that of standard ones. Can I bias the coin, so that a cascade is **more** than 50% likely, to achieve the effect of cascading after (say) one and a half children are cut? Can this be used to improve the time for delete-min at the cost of increasing the time for decrease key?