Problem Set 1

Due: September 21, 1999.

Starred (*) questions are likely to be more challenging.
Double starred (**) questions are for brownie points.

Problem 1. Professor Pinocchio claims that the height of an n-node Fibonacci heap is $O(\log n)$. Show that the Professor is mistaken by exhibiting, for any positive integer n, a sequence of Fibonacci heap operations that creates a Fibonacci heap consisting of just one tree that is a linear chain of n nodes.

Problem 2. In this problem we'll try to understand more about the constant factors associated with Fibonacci heaps. Assume for this problem that single "node operations" such as cutting a node, linking a node as a root or a child, checking a mark bit, or comparing two keys, take one time unit. In-register computations involving a constant number of items are free (you can tinker with this model as you need to; just be explicit).

(a) Tune the potential function to get the best bounds (measure constant factors) on the amortized number of such basic operations per heap operation.

(b) Suppose we change Fibonacci heaps so that up to 4 children can be cut from a node before we cascade the cut. Which operations get faster (by constant factors) and which slower? By how much?

Problem 3. More on amortizing heap operations. The bounds asked for can of course be derived from Fibonacci heaps; the goal is to work with the data structures specified rather than introducing complicated new ones.

(a) Let P be a priority queue that performs insert, delete-min and merge in $O(\log n)$ time and 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 amortized $O(1)$ time, without affecting the cost of delete-min or merge. Assume that the priority queue does not support an efficient decrease-key.

(b) Using the above technique, show that even binary heaps can be modified to achieve amortized constant time insert operation and $O(\log n)$ delete-min. Note that binary heaps do not support $O(\log n)$ merge operation.

Problem 4. In the multi-level buckets data structure, deletions are expensive because we may have to search though many empty buckets to find the minimum item. Suppose that at each level we keep a Fibonacci heap indicating which buckets are occupied. Note this is **not** the "heap-on-top" queue from "Buckets, Heaps, Lists, and Monotone Priority Queues."

(a) What tradeoff in time bounds does this give for insert, delete-min, and decrease-key?

(b) What is the best running time you can achieve for shortest paths using this modified structure?

Problem 5. In class we saw how to use a van Emde Boas priority queue to get $O(\log\log U)$ time per queue operation (insert, delete-min, decrease-key) when the range of values is U. Show that for shortest paths (on a graph with n nodes and range of edge lengths C), although the range of values is nC, we can get $O(\log\log C)$ time per queue operation.

* **Problem 6.** Augment the VEB priority queue to support the following operations on integers in the range $1 \mathinner{..} U$ in $O(\log\log U)$ time each:

find(v) report whether the item v is stored in the structure

pred(v) return v's predecessor, the item of largest value less than v (return nothing if v is the minimum item)

succ(v) return v's successor, the item of smallest value greater than v (return nothing if v is the minimum item)

** **Problem 7.** 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?

** **Problem 8.** Can a VEB type data structure be combined with some ideas from Fibonacci Heaps to support insert/decrease-key in $O(1)$ time with delete-min in $O(\log\log U)$ time?