Problem Set 6

This problem set is due in lecture on Monday, November 6, 2005. The homework lab for this problem set will be held 2–4 P.M. on Sunday, November 5, 2006, in room 32-124.

Reading: Chapters 17 and the handout on competitive analysis.

Both exercises and problems should be solved, but only the problems should be turned in. Exercises are intended to help you master the course material. Even though you should not turn in the exercise solutions, you are responsible for material covered in the exercises.

Mark the top of each sheet with your name, the course number, the problem number, your recitation section, the date and the names of any students with whom you collaborated. Please staple and turn in your solutions on 3-hole punched paper.

You will often be called upon to “give an algorithm” to solve a certain problem. Your write-up should take the form of a short essay. A topic paragraph should summarize the problem you are solving and what your results are. The body of the essay should provide the following:

1. A description of the algorithm in English and, if helpful, pseudo-code.
2. At least one worked example or diagram to show more precisely how your algorithm works.
3. A proof (or indication) of the correctness of the algorithm.
4. An analysis of the running time of the algorithm.

Remember, your goal is to communicate. Full credit will be given only to correct solutions which are described clearly. Convoluted and obtuse descriptions will receive low marks.

---

Exercise 6-1. Do Exercise 17.1-1 on page 410 in CLRS.

Exercise 6-2. Do Exercise 17.3-3 on page 416 in CLRS.

Exercise 6-3. Do Exercise 17.3-6 on page 416 in CLRS.

Exercise 6-4. Do Exercise 17.3-7 on page 416 in CLRS.

Exercise 6-5. Do Exercise 17.4-2 on page 425 in CLRS.
Problem 6-1. Corporate Chaos

You are a qualitative analyst for a financial institution trying to keep track of who’s who in today’s corporate frenzy of startups, mergers, and acquisitions. Every person in your database is employed by exactly one company. Your job is to implement a data structure that, given such person \( x \), will produce the name of the president of the company employing person \( x \). More precisely, your data structure must efficiently support the following three operations:

1. \textsc{Startup}(x) creates a new company whose president and only employee is person \( x \) (previously unemployed).
2. \textsc{Merger}(x, y) merges the two (distinct) companies employing persons \( x \) and \( y \) respectively.
   The president of the resulting company is one of the presidents of the two subcompanies, and
   the data structure gets to choose which (!).
3. \textsc{President}(x) returns the president of the company employing person \( x \).

You first idea is to represent each company as a doubly linked list, with each node corresponding to a person, where the head of the list stores the president, and each node \( x \) stores a pointer \( \text{head}[x] \) to the head of the list. \textsc{President}(x) simply returns \( \text{head}[x] \), in \( O(1) \) time. \textsc{Merger}(x, y) appends \( y \)’s list to the end of \( x \)’s list, and then changes all of the \( \text{head} \) pointers in \( y \)’s list to \( \text{head}[x] \).

(a) Give a tight asymptotic bound on the worst-case total cost of performing \( m \) \textsc{Startup}, \textsc{Merger}, and/or \textsc{President} operations (starting from an empty data structure).

You think of a simple trick to improve the performance of your data structure. You augment the head \( h \) of each list with two pieces of information: a counter \( \text{length}[h] \) tracking the length of the list, and a pointer \( \text{tail}[h] \) to the tail of the list. Now, \textsc{Merger}(x, y) always appends the smaller of the two lists to the longer of the two lists (breaking ties arbitrarily), updating the \( \text{head} \) pointers in \( y \)’s list to \( \text{head}[x] \).

(b) Prove that the worst-case total cost of performing \( m \) \textsc{Startup}, \textsc{Merger}, and/or \textsc{President} operations (starting from an empty data structure) is \( \Theta(m \log m) \).

Somewhat unhappy about the lackluster performance of your \textsc{Merger} implementation when merging large companies, you decide to modify your data structure so that each company is represented by a (not necessarily binary) tree rather than a list. Each node \( x \) represents a person, and stores a pointer \( \text{parent}[x] \) to its parent; however, nodes do not store pointers to their children.

The root of a tree represents the president of the corresponding company. To find \textsc{President}(x), we follow the path from \( x \) to the root of the tree by repeatedly following \( \text{parent} \) pointers, paying a cost proportional to the depth of \( x \). \textsc{Merger}(x, y) makes \textsc{President}(y) a new child of \textsc{President}(x) by setting \( \text{parent}[\text{President}(y)] \leftarrow \text{President}(x) \). (Note that we do not try to make the smaller tree a subtree of the larger tree.)

You decide to adopt a new trick which might improve the performance of your data structure. When calling \textsc{President}(x_d) on a node of depth \( d \geq 2 \), let \( x_d, x_{d-1}, \ldots, x_0 \) denote the sequence
of nodes encountered on the path from $x_d$ to the root $x_0$ of the tree. Without asymptotically increasing the $\Theta(d)$ cost of the operation, you can set $parent[x_i] \leftarrow x_0$ for $i = 1, 2, \ldots, d$. Note that the two calls to PRESIDENT made during MERGER also use this trick.

(c) Show that the total cost of performing $m$ consecutive PRESIDENT operations on a set of companies with a total number of $n$ people is $O(n + m)$.

(d) Show that the total cost of performing $m$ STARTUP, MERGER, and/or PRESIDENT operations (starting from an empty data structure) is $O(m \log m)$.

Hint: Use the following potential function of a forest:

$$\Phi = c \sum_{\text{node } x} \log w(x),$$

where the weight $w(x)$ of a node $x$ is the number of nodes in the subtree rooted at $x$ (including $x$ itself), and $c$ is some positive constant.

### Problem 6-2. Competitive Corruption

**Note on “weak” vs. “strict” competitiveness.** In lecture, a $c$-competitive online algorithm was defined to have cost at most $c$ times that of the optimal offline algorithm, plus an additive constant $k$. This is technically the definition of a weakly $c$-competitive algorithm. If $k = 0$, then we say that the online algorithm is strictly $c$-competitive. For the purposes of this problem, you should assume that “competitive” means “strictly competitive”. (This assumption will make your solutions easier. One could actually prove the same results in terms of weak competitiveness, but this would complicate parts (b) and (d).)

A country (best left unnamed) is so rife with corruption that the only way to deal with government officials is through bribery. Unfortunately, it is considered highly impolite to suggest that someone may be bribed, and even more so to ask outright “How much . . .”. Instead, one must guess the amount of money that will make an official accede to a given request (assume this is always at least 1 unit of the local currency), and then quietly slip the money under the table. If the bribe is deemed sufficient, the official accedes to the request. If not, the official pockets the money but acts as if nothing has happened, and one has to try again with a larger bribe. Note that multiple, unsuccessful small bribes do not “add up” in the eyes of an official, who only considers the “current” bribe (while also pocketing all previous bribes).

(a) Show that there exists a deterministic $c$-competitive bribing strategy, i.e., a strategy that, for some $c > 0$, will pay at most $c$ times the minimum amount $b$ an official would deem a sufficient bribe, whatever that amount is.

(b) Prove that any deterministic strategy has a competitive ratio of at least 4. (Hint: what would otherwise happen to the ratio between the sum of the first $i$ bribes and the sum of the first $i - 1$?)

(c) Can a randomized strategy do better?
(d) Often, there are several different officials that one can bribe to achieve a certain goal. Suppose one only needs bribe one of $k$ officials (who may each have a different “price”). Prove the best possible competitive ratio of a deterministic strategy is $\Theta(k)$. (In particular, why doesn’t it suffice to just bribe the first official using a 4-competitive strategy?)