Problem Set 6 Solutions

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. STARTUP($x$) creates a new company whose president and only employee is person $x$ (previously unemployed).
2. 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. 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 $head[x]$ to the head of the list. PRESIDENT($x$) simply returns $head[x]$, in $O(1)$ time. MERGER($x, y$) appends $y$’s list to the end of $x$’s list, and then changes all of the head pointers in $y$’s list to $head[x]$.

(a) Give a tight asymptotic bound on the worst-case total cost of performing $m$ STARTUP, MERGER, and/or PRESIDENT operations (starting from an empty data structure).

Solution: $\Theta(m^2)$.

More precisely, it’s certainly $O(m^2)$ because, after at most $m$ operations, the length of any list (and thus the cost of MERGER) is at most $O(m)$. It can also be $\Omega(m^2)$: if we maintain a single company “grown” through $m/2$ calls to STARTUP, each followed by a call to MERGER that appends the company’s employee list to the length-1 employee list of the new startup, we pay at least $\Omega(m/4)$ for each of the last $m/4$ calls to MERGER.

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 $length[h]$ tracking the length of the list, and a pointer $tail[h]$ to the tail of the list. Now, MERGER($x, y$) always appends the smaller of the two lists to the longer of the two lists (breaking ties arbitrarily), updating the head pointers in the smaller list.
(b) Prove that the worst-case total cost of performing $m$ STARTUP, MERGER, and/or PRESIDENT operations (starting from an empty data structure) is $\Theta(m \lg m)$.

**Solution:** $\Theta(m \lg m)$.

Let us first prove that the total cost can be $\Omega(m \lg m)$. Let $m = 2^b$. Then, creating $2^{b-1}$ startups requires $2^{b-1}$ STARTUP operations; and recursively merging $2^{b-i-1}$ companies of size $2^i$ into half that many companies of twice the size requires $2^{b-i-2}$ MERGER operations, each with a cost of $2^i$. The total number of operations performed is then $\sum_{i=1}^{b} 2^{b-i} = 2^b - 1 = m - 1$, and the total cost is $\Omega(\sum_{i=0}^{b-1}(2^{b-i-2} \cdot 2^i)) = \Omega(b \cdot 2^{b-2}) = \Omega(m \lg m)$.

Let us now prove that the total cost of adjusting nodes in MERGER operations is $O(m \lg m)$; this proves the thesis since the cost of PRESIDENT and STARTUP, as well as the cost of adjusting the length counter in MERGER, is $O(1)$. A node is adjusted (at $O(1)$ cost) only when its current list is concatenated to one at least as large - i.e. when the size of the node’s list at least doubles. This can happen at most $\lg m$ times for each node, and thus (since $m$ operations yield at most $m$ nodes) the total cost of all MERGER operations is at most $m \lg m$.

Somewhat unhappy about the lackluster performance of your 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 PRESIDENT($x$), we follow the path from $x$ to the root of the tree by repeatedly following parent pointers, paying a cost proportional to the depth of $x$. MERGER($x, y$) makes PRESIDENT($y$) a new child of 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 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 $\text{parent}[x_i] \leftarrow x_0$ 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 $O(n + m)$.

**Solution:** Define the potential of a forest be the number of nodes at depth greater than 1, which is clearly at most $n$ for a forest of $n$ nodes. A PRESIDENT($x$) operation called on a node $x$ at depth $d > 1$—call such an operation expensive—reduces the potential of the forest by $\Omega(d)$, at a cost of $O(d)$. Thus, the total cost of expensive PRESIDENT operations is $O(n)$, while the total cost of all other $O(m)$ PRESIDENT operations (called on nodes at depth $O(1)$ and thus costing $O(1)$ each) is $O(m)$.
(d) Show that the total cost of performing $m$ STARTUP, MERGER, and/or PRESIDENT operations (starting from an empty data structure) is $O(m \lg m)$.

*Hint:* Use the following potential function of a forest:

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

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.

**Solution:** Clearly, during the first $m$ operations the forest of companies always has size at most $m$. Recall that there exists some fixed constant $c$ such that the cost of a STARTUP operation is at most $c/2$ and the cost of a MERGER or PRESIDENT operation involving nodes at depth at most $d$ is at most $d \cdot c/2$. Let $C_i$ be the cost of the $i^{th}$ operation, and let $\Phi_i$ be the potential of the forest after $i$ operations (equal to the sum of $c \lg w(x)$ for every node $x$ in the forest - note that $\Phi_0 = 0$ and $\Phi_i \geq 0$).

A STARTUP operation does not increase the potential of the forest, since it adds a node of weight 1 and $c \lg 1 = 0$. A MERGER or PRESIDENT operation can at most increase the weight of one (root) node, and by at most $m$, which increases the potential of the forest by at most $c \lg m$.

Let $A_i = C_i + (\Phi_i - \Phi_{i-1})$. Call a node heavy if its weight is at least half that of its parent and light otherwise (a root is light). If the $i^{th}$ operation involves nodes at depth $d < 2 \lg m$, then $A_i \leq d \cdot c/2 + c \lg m < 2c \lg m$. If it involves nodes at depth $d \geq 2 \lg m$, at least $d/2$ of those nodes will be heavy. Unlinking a heavy node from its parent (to link it to the root) reduces the weight of the parent by at least a factor 2, and therefore reduces the contribution of that parent to the potential of the forest by at least $c \lg 2 = c$. In this case, too, $A_i \leq d \cdot c/2 + c \lg m - c \cdot d/2 \leq 2c \lg m$.

Therefore $\sum_{i=1}^{m} (C_i) \leq \sum_{i=1}^{m} (C_i) + (\Phi_m - \Phi_0) = \sum_{i=1}^{m} (C_i + (\Phi_i - \Phi_{i-1})) = \sum_{i=1}^{m} A_i \leq m \cdot 2c \lg m = O(m \lg m)$, and the amortized cost per operation is $O(\lg m)$.

[If you’re interested in what happens when you combine the size trick with the parent trick, read Chapter 21 of CLRS.]

**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.

Solution: Begin offering the official a bribe of 1 (for brevity, all amounts of money are assumed to be in units of the local currency from now on), then keep doubling the bribe until the official accedes to the request. To prove that this strategy has a competitive ratio no higher than 4, let \( b_{min} \) be the minimum sufficient bribe, \( i \) be the least \( i \) such that \( 2^i \geq b_{min} \), and \( b_{tot} \) be the total amount spent in bribes. Then \( b_{tot} = \sum_{i=0}^{t} 2^i = 2^{i+1} - 1 < 4 \cdot 2^{i-1} < 4 \cdot b_{min} \).

(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 \( i - 1 \)?)

Solution: Let \( b_i \) be the \( i^{th} \) bribe after bribes \( b_1, \ldots, b_{i-1} \) have proved unsuccessful; let \( \sigma_i = \sum_{j=1}^{i} b_j \); and let \( \rho_i = \sigma_i / \sigma_{i-1} \geq 1 \). If the “price” of an official is \( b_i + \delta \) for \( 0 < \delta \leq b_{i+1} - b_i \), then the ratio between the total amount paid and the minimum sufficient bribe is \( \sigma_{i+1} / (b_i + \delta) \); and therefore the competitive ratio is equal to the supremum, over \( i \), of \( \frac{\sigma_{i+1}}{b_i} \). Assume by contradiction the existence of a \((4 - \epsilon)\) competitive deterministic strategy for some positive \( \epsilon \leq 3 \). Then, if \( \rho_i > 0 \), we would have \( \rho_{i+1} \leq (4 - \epsilon)(1 - 1 / \rho_i) < \rho_i - \epsilon / 4 \), where the second inequality is satisfied since \(((\rho_i - \epsilon / 4) - ((4 - \epsilon)(1 - 1 / \rho_i)))\rho_i = \rho_i^2 - (4 - 3\epsilon / 4)\rho_i + (4 - \epsilon) > 0 \). This would imply \( \rho_i \leq 0 \) for some \( i \).

(c) Can a randomized strategy do better?

Solution: A randomized strategy can have a competitive ratio lower than 4. Flip a coin and set the first bribe \( b_1 \) to 1 if heads and to \( \sqrt{2} \) if tails; set the \((i + 1)^{th}\) bribe \( b_{i+1} \) (if the \( i^{th}\) bribe \( b_i \) was unsuccessful) to \( 2^j \cdot b_1 = (\sqrt{2})^{2^j} \cdot b_1 \). If the minimum amount that will make the official accede to a request is \((\sqrt{2})^n / m \) with \( 1 \leq m < \sqrt{2} \), then the first successful bribe with probability \( 1/2 \) is \( m \) times that amount, and with
probability $1/2$ is $\sqrt{2}m$ times that amount. Since every bribe is twice the previous one, the ratio between the total amount paid and the minimum sufficient amount is then (in expectation) no more than $(\frac{1}{2}m + \frac{1}{2}\sqrt{2}m) \sum_{i=0}^{\infty}(\frac{1}{2})^i \leq 2 + \sqrt{2} < 4$.

(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?)

Solution: Performing the 4-competitive strategy from part (a) “in parallel” for all $k$ officials (first paying 1 to each, then 2 etc.) clearly results in a $4k$ competitive strategy. We now prove that no deterministic bribing strategy can be better than $\Theta(k)$ competitive. Any competitive bribing strategy must eventually attempt to bribe each official at least once, since that official’s price could be arbitrarily lower than the others’. After every official has been paid once, let $\sigma^-$ and $\sigma^+$ be, respectively, the least and largest total amount paid to an official. For any $\epsilon > 0$, if the price of one official who has received only $\sigma^-$ is $(1+\epsilon)\sigma^-$ and that of all others is $(1+\epsilon)\sigma^+$, no bribe has been successful even after paying (in total) at least $\frac{k}{1+\epsilon}$ times the minimum price of an official.