Selected Solutions for Chapter 21: Data Structures for Disjoint Sets

Solution to Exercise 21.2-3

We want to show that we can assign $O(1)$ charges to MAKE-SET and FIND-SET and an $O(\lg n)$ charge to UNION such that the charges for a sequence of these operations are enough to cover the cost of the sequence—$O(m + n \lg n)$, according to the theorem. When talking about the charge for each kind of operation, it is helpful to also be able to talk about the number of each kind of operation.

Consider the usual sequence of m MAKE-SET, UNION, and FIND-SET operations, n of which are MAKE-SET operations, and let $l < n$ be the number of UNION operations. (Recall the discussion in Section 21.1 about there being at most $n - 1$ UNION operations.) Then there are n MAKE-SET operations, l UNION operations, and $m - n - l$ FIND-SET operations.

The theorem didn't separately name the number l of UNIONs; rather, it bounded the number by n. If you go through the proof of the theorem with l UNIONs, you get the time bound $O(m - l + l \lg l) = O(m + l \lg l)$ for the sequence of operations. That is, the actual time taken by the sequence of operations is at most $c(m + l \lg l)$, for some constant c.

Thus, we want to assign operation charges such that

$$
\begin{array}{lcl}
(\text{MAKE-SET charge}) & \cdot & n \\
+(\text{FIND-SET charge}) & \cdot & (m - n - l) \\
+(\text{UNION charge}) & \cdot & l \\
\hline
\multicolumn{3}{l}{\geq c(m + l \lg l)\ ,}
\end{array}
$$

so that the amortized costs give an upper bound on the actual costs.

The following assignments work, where c' is some constant $\geq c$:

- MAKE-SET: c'
- FIND-SET: c'
- UNION: $c'(\lg n + 1)$

Substituting into the above sum, we get

$$
\begin{aligned}
c'n + c'(m - n - l) + c'(\lg n + 1)l &= c'm + c'l \lg n \\
&= c'(m + l \lg n) \\
&> c(m + l \lg l)\ .
\end{aligned}
$$

Solution to Exercise 21.2-6

Let's call the two lists A and B, and suppose that the representative of the new list will be the representative of A. Rather than appending B to the end of A, instead splice B into A right after the first element of A. We have to traverse B to update pointers to the set object anyway, so we can just make the last element of B point to the second element of A.