CS170 Problem Set 6 Out: Feb 13, 2001
Due: Feb 23, 2001

1. Using a forest-based disjoint sets data structure with path compression and union-by-rank, a sequence of n MakeSet, Link, and Find operations can take asymptotically slightly more than linear time in the worst case. (Note we have omitted Union operations. Assume that Link operations are done only on set roots.)

   (a) Explain why if all the Finds are done before all the Links, a sequence of n operations is guaranteed to take $O(n)$ time.

   (b) Explain why if all the Links are done before all the Finds, a sequence of n operations is guaranteed to take $O(n)$ time. Note that a single Find operation does **not** necessarily take $O(1)$ time, so you will have to show that the complete sequence of operations takes only linear time even if some of the operations take greater than constant time. (Expect this to be a difficult question. Hint: what is the relationship between the number of grandchildren in the disjoint sets data structure and the running time of the Find operations?)

   (c) Suppose we use the disjoint sets data structure *without* path compression (but we still use union-by-rank). We begin by running n MakeSet operations. Following that, give a sequence of m Link and Find operations that require $\Omega(m \log n)$ steps, where $m \geq 2n$.

2. Suppose we want to augment the basic forest-based disjoint sets data structure with a Remove(i) operation that removes an item i from the set that contains it, and moves it into a separate set of its own. All the other items that were in the same set as i are still in a single set. Although we are supporting this new operation, we require that a sequence of m Union and Find operations (with no Remove operations) must still take $O(m \cdot \alpha(m, n))$ time, where n is the number of items (i.e. the number of MakeSet operations).

   (a) Describe an $O(n)$ algorithm for Remove(i) that uses the forest-based data structure with no modifications (e.g. no extra pointers). Make sure you have considered all possible cases. Don't worry about correcting any node's rank until part (b), next.

   (b) To complete your answer to part (a), describe an $O(n)$-time algorithm that will assign each node in the disjoint sets data structure a rank equal to its height in the tree (i.e. the depth of its deepest descendant). You may use an auxiliary data structure.

   (c) Assuming that you may make changes to the disjoint sets data structure, can Remove(i) be implemented in $O(s)$ time, where s is the size of the set containing the item i? If so, explain how (including how to update the rank fields). If not, explain why.

3. CLR Problem 22-2.