CS 181b
Advanced Topics in Algorithms
Spring 2002
Problem Set 2b
Due Tuesday, February 5 in class

1. **[10 Points] Minimum Spanning Trees.** This problem will simply demonstrate one of the uses of the union-find data structure. Consider the problem of finding a minimum spanning tree in a graph with n vertices and $O(n)$ edges where each edge has an integer weight between 0 and n. Show that a minimum spanning tree can be found in time $O(n \log^* n)$. You may use your CS 140 algorithms textbook.

2. **[15 Points] Degenerate Splay Trees.** The objective of this problem is to give you a chance to play with the splaying rules. You may wish to use a splay tree simulation tool (such as the one on the course web page) to help you. Describe a sequence of splay tree operations such that at the end of these operations the splay tree contains n nodes where n is odd **and** the tree is a path which zig-zags at each level. That is, the nodes alternate between being left and right children.

3. **[15 Points] Analyzing Insertion into Splay Trees.** In class we analyzed the amortized cost of performing a FIND in a splay tree. We showed that while the actual cost of the FIND is d (where d is the depth of the last node visited during the FIND), the change in potential due to the subsequent splaying step was less than or equal to $(3 \log_2 n) - d + 2$, giving us an amortized cost of $(3 \log_2 n) + 2 \in O(\log_2 n)$.[1]

   Now let's find the amortized cost of performing an INSERT in a splay tree. The actual cost of the insertion and subsequent splaying is d, the depth of the leaf at which the insertion is performed. However, the amortized cost will be less because of the potential change. The potential changes in two ways. First the potential changes because a node is inserted at the bottom of the tree, increasing the rank, $r(v)$, of each node on the path from the root to this leaf. Next, the potential changes due to the splay. We already know that the splay operation contributes a change of potential which is less than or equal to $(3 \log_2 n) - d + 2$. So, we only need to determine the change in potential due to the increase in ranks along the insertion path.

   Let $n(v)$ and $r(v)$ denote the number of nodes in the tree rooted at v and the rank of v, respectively, before the insertion. Let $n'(v)$ and $r'(v)$ denote these values immediately after the insertion (but before the splay). Show that the change in the tree's potential immediately after the insertion but before the splay is $O(\log_2 n)$. (Note: To do this, write down the simple relationships that you can find between the various n, r, n' and r' variables. Then formulate a summation for the change in potential. Finally, find an

[1] You might argue that the actual cost of a FIND is $2d$ since we first pay d to go down the tree and then pay d again to do the splay step. In reality, the cost is some constant K times d and we just use 1 for this constant to keep life simple. If we want to keep the constant K around (because living simply is boring), we can just scale the potential function up by that same constant so that the Kd real cost is now offset by a Kd decrease in the potential. Of course, now the $3 \log_2 n$ term now becomes $3K \log_2 n$.

upper-bound on this summation.) Now put everything together to conclude that the amortized cost of an INSERT in a splay tree is $O(\log_2 n)$.

4. **[15 Points] Union-Find with Partial Path Compression.** In class we examined the union-find data structure using union-by-size and path compression. One disadvantage of path compression is that it is a two-phase process: First we must "climb" the path to find the root. Then, we go back and set the parent of each node to be the root.

   A simpler approach is to use *partial path compression* in which each node on the path has its parent pointer changed to point to its grandparent. This process can be done in just one pass as we climb up the path. Show that using partial path compression instead of full path compression still allows us to perform any sequence of n UNION and FIND operations in time $O(n \log^* n)$ time.

5. **[15 Points] Union-Find Again.** Consider again the union-find data structure using union-by-size and path compression. Assume that we will perform a sequence of n UNION and FIND operations where all of the UNION's are done first and then we do all of the FINDs. Show that the total running time is now asymptotically even better than $\Theta(n \log^* n)$.