Homework #11
Introduction to Algorithms/Algorithms 1
600.363/463
Spring 2014

Due on: Tuesday, April 22nd, 5pm
Late submissions: will NOT be accepted
Format: Please start each problem on a new page.
Where to submit: On blackboard, under student assessment
Please type your answers; handwritten assignments will not be accepted.
To get full credit, your answers must be explained clearly,
with enough details and rigorous proofs.

April 15, 2014

1 Problem 1 (20 points)

(a) (10 points) Suppose we have a set of n vertices $V = \{v_1, v_2, v_3, \ldots, v_n\}$. and that we have two binary trees $T_1 = (V, E_1)$ and $T_2 = (V, E_2)$. Prove that we can make tree T_1 "look like" tree T_2 using $O(n)$ rotations (i.e., some mix of left- and right-rotations). What me mean by a tree T "looking like" another tree T' is that they are the same up to a relabeling of the nodes– so, for example, the root of T might be vertex v_i and the root of tree T' might be vertex v_j, but if we ignored the subscripts on the vertices, T and T' are the same tree. Put another way, $T = (V, E)$ "looks like" $T' = (V, E')$ if there exists a permutation $\pi : V \mapsto V$ such that $E' = \{(\pi(a), \pi(b)) : (a, b) \in E\}$.

Hint: first, show that we can turn tree T_1 into a linear chain via some sequence of $O(n)$ rotations. Then show that we can turn a linear chain into a tree of any "shape" we please via a sequence of $O(n)$ rotations. Specifically, we'll be able to turn the linear chain into a tree with the same "shape" as T_2, i.e., we'll make a tree that "looks like" T_2 in the sense described above. You may find it useful here to describe trees (and subtrees) in terms of two numbers– the number of

nodes that are descendants of the left child of the root and the number of nodes that are descendants of the right child of the root.

(b) (10 points) Prove that red-black trees are indeed balanced by showing that for any two simple paths P_1 and P_2 from the root to a leaf, we have $\frac{1}{2}|P_2| \leq |P_1| \leq 2|P_2|$, where $|P|$ denotes the length of path P.

2 Problem 2 (20 points)

(a) (5 points) Prove that for any node a in a disjoint-set forest (also known as Union-Find with path-compression) the rank of a is less than or equal to the rank of $a.p$, with inequality when $a \neq a.p$.

(b) (5 points) Suppose we have a disjoint-set forest (that is, the data structure for Union-Find with path compression) on n elements. Prove that every node in the disjoint-set forest has rank $O(\log n)$.

(c) (10 points) In a previous homework, you were asked to give an algorithm that took as input an undirected graph $G = (V, E)$ and determined whether or not G contained a cycle. Let's revisit that problem from a different angle. As before, we would like to determine whether or not the graph $G = (V, E)$ is a forest. It is likely that your algorithm the first time you had to solve this problem used union-find (or something similar) implicitly as a black box. Devise an algorithm that takes as input a graph $G = (V, E)$ and returns `True` if G is a forest and returns `False` otherwise. Put another way, your algorithm should return `False` if the input graph $G = (V, E)$ contains a cycle, and `True` otherwise. Your algorithm should make explicit use of the union-find data structure and should run in (amortized) $O(|V| + |E| + |E|\alpha(|V|))$ time, where α is defined to be

$$\alpha(n) = \min\{k : A_k(1) \geq n\},$$

and where $A_k(j)$ is the Ackermann function, defined in CLRS on page 573 (section 21.4) (note that the $|V| + |E|$ term in the runtime is simply the time that we need to read the input).

Prove the correctness and the runtime of your algorithm.

Optional exercises

Solve the following problems and exercises from CLRS: 21.3-2, 21.3-3, 13.1-6, 13.1-4, 13.1-3, 17.1-3, 17.2-2, 17.3-2.