COS 528 Problem Set 1: Union-Find Due: March 13, 2013
Spring 2013
Tarjan

Do not consult outside references other than class materials. You may discuss these problems with your classmates, but write the solutions entirely on your own. For full credit, do either Problem 1 or Problem 2, and either Problem 3 or Problem 4.

1. (Efficiency of path halving) Consider the version of union-find that uses path halving (make each node on the find path point to its grandparent) instead of path compression, and linking by rank. Prove that the amortized cost per find is $O(\alpha(n, d))$. Hint: Define levels and indices as in the upper bound argument in the slides and done in class on 2/18. Consider the levels of vertices along a find path. Let x, y, and z be successive vertices along the path, so that y is the parent of x before the halving and z is its parent after. Let k, k' denote levels before and after the find, respectively. Prove that if $k(y) > k(x)$, then $k'(x) >= k(y)$. That is, the halving increases $k(x)$ by at least $k(y) - k(x)$. Prove that if $k(y) = k(x)$, then either $k'(x) > k(x)$, or $k'(x) = k(x)$ and the halving strictly increases the index of x. Using these results, find a way to charge each node on a find path either to an increase in the level or index of some node, or to the find, so that the charge to each find is at most $\alpha(n, d)$.

Optional: Prove the same bound for path splitting (after a find, make every other node along the find point to its grandparent). Hint: Define the level and index of a node as a function of its rank and the rank of its grandparent, not that of its parent. Find a way to charge each node on the find path to the find, or to an increase in the level or index of a node, so that only a constant number of nodes are charged to each increase, and the find incurs a charge of at most $\alpha(n, d)$.

2. (Analysis of Rem's algorithm) Consider the following interesting algorithm proposed by Rem to solve the variant of the union-find problem in which there are n elements, each initially in a singleton set, and the only operation is *contingent union*, defined as follows:

c-unite(x, y): if elements x and y are in the same set, return **false**; otherwise, unite the sets containing x and y and return **true**.

Rem's implementation assumes that the elements are uniquely labeled from 1 to n, so that the elements are totally ordered by label. (That is, $x < y$ if the label of x is less than the label of y.) The implementation represents each set by a rooted tree whose nodes are the elements in the set, with the largest node being the root and every other node having a parent larger than itself. To simplify the implementation, each root is its own parent. The implementation does *c-unite*(x, y) by concurrently following parent pointers from x and y, splicing the two find paths together, until reaching a common node or reaching a tree root:

c-unite(*x*, *y*):
  while true do
    if *p*(*x*) = *p*(*y*) **then return false**
    else
      {**if** *p*(*x*) < *p*(*y*) then *x* ↔ *y* **fi**;
       z ← *p*(*y*);
       p(*y*) ← *p*(*x*);
       if *y* = *z* **then return true else** *y* ← *z*}
    fi
  od

Compare this to the find-based implementation of *c-unite*:

c-unite(*x*, *y*):
  v ← *find*(*x*); *w* ← *find*(*y*);
  if *v* = *w* **then return false else** {*link*(*v*, *w*); **return true**} *fi*

Whereas the latter only links roots to roots, the former can link a root to an arbitrary node.

Prove that the amortized time per contingent union is $O(\log_{(d+1)} n)$ in the worst case (the worst possible element numbering), where n is there number of elements, m is the number of contingent unions, and $d = \lceil m/n \rceil$.

3. (From Seidel: getting the inverse Ackermann bound directly from the Seidel-Sharir recurrence) Let $f(m, n, r)$ be the maximum cost (number of pointer changes) of performing a sequence of m path compressions on a union tree built by linking by rank having n nodes and maximum rank r.

For integers $k, j \geq 0$ define

$R(k, j) = \max\{r \mid f(m, n, r) \leq km + jn \text{ for all } m, n \geq 0\}$.

Prove that R satisfies

$R(k, 0) = k + 1$
$R(0, j) = j + 1$
$R(k, j) \geq R(k, j - 1) + R(k - 1, R(k, j - 1) + j - 1) + 1 \geq R(k - 1, R(k, j - 1))$

What properties of ranks do you need to obtain this result?

4. (Path compression with random linking) Consider a union-find problem in which the n elements are labeled randomly from 1 to n (with each permutation equally likely), and each link makes the root of larger label the parent of the root of smaller label. Define the rank of a node to be its height in the union tree (the tree build by

the links, ignoring any compressions). Prove that the expectation of the sum of the node ranks is $O(n)$, where the expectation is taken over the possible numberings of the elements, each numbering being equally likely. Use this to prove that the expected time for m finds is $O(m\alpha(n, d))$. Hint: The expected time bound follows immediately from the bound on the expected sum of node ranks using the analysis of path compression with linking by rank presented in class.

Research Problem: Consider Rem's contingent union algorithm (Problem 2). Suppose that, for a particular sequence of contingent unions, the elements are labeled randomly from 1 to n, with all permutations equally likely. (The splicing depends on the order imposed by the labeling.) Prove or disprove that the expected sum of heights in the union tree is $O(n)$. A positive result would imply that the random version of Rem's algorithm takes $O(\alpha(n, d))$ expected amortized time per contingent union. The difficulty here (as compared to problem 4) is that the algorithm links roots to arbitrary nodes, not just to other roots. Thus a link can increase the heights of many nodes in the current forest (all the ancestors of the node getting a new child).