CS 681 Fall 2001 – Midterm

The midterm is due at 2:30 pm on Friday, October 26, in class. You can contact me (either by e-mail or in person) or Niranjan with any questions; you are not allowed to discuss the exam with anyone else.

(1) Suppose we are given a directed network $G = (V, E)$ with a root node r, and a set of *terminals* $T \subseteq V$. We'd like to disconnect many terminals from r, while cutting relatively few edges.

We make this trade-off precise as follows. For a set of edges $F \subseteq E$, let $q(F)$ denote the number of nodes $v \in T$ such that there is no r-v path in the subgraph $(V, E - F)$. Give a polynomial-time algorithm to find a set F of edges that maximizes the quantity $q(F) - |F|$. (Note that setting F equal to the empty set is an option.)

(2) Recall the implementation of the *Union-Find* data structure from lecture and Kozen's book (chapters 10 and 11). We have a universe of n elements, labeled $\{1, 2, \ldots, n\}$. Initially, each element forms its own single-node tree $T_i = \{i\}$. The operation $Union(T_a, T_b)$ simply links the root of the smaller tree among T_a, T_b to the root of the larger, producing a single tree. Thus, a single $Union$ operation takes constant time. The operation $Find(i)$ follows the path from i to the root r of the tree containing it. We will be considering the version of the $Find$ operation that is optimzed to use *path compression*: after the path from i to r is followed, all elements on this path are updated to point directly to the root r. A single $Find$ operation (with path compression) takes time proportional to the length of the path from i to r.

Now consider the following alternative implementation of a *Union-Find* data structure. We have trees just as before, but now the operation $Union(T_a, T_b)$ links the root of T_a to the root of T_b, *regardless of which tree is larger*. (Again, this takes constant time.) The operation $Find(i)$ is as described above; it is implemented to use path compression.

Under this new implementation of $Union$ and $Find$, is it possible for a sequence of n interleaved $Union$ and $Find$ operations to take $\Omega(n^2)$ time? In other words, is there a constant $\varepsilon > 0$ so that for arbitrarily large values of n, there are sequences of n interleaved $Union$ and $Find$ operations that take time $\geq \varepsilon n^2$? Give a proof to support your answer.

(3) Let's go back to the original motivation for the minimum spanning tree problem: we are given a connected, undirected graph $G = (V, E)$ with positive edge lengths $\{\ell_e\}$, and we want to find a spanning subgraph of it. Now, suppose we are willing to settle for a subgraph $H = (V, F)$ that is "denser" than a tree, and we are interested in guaranteeing that for each pair of vertices $u, v \in V$, the length of the shortest u-v path in H is not much longer than the length of the shortest u-v path in G. By the *length* of a path P here, we mean the sum of ℓ_e over all edges e in P.

Here's a variant of Kruskal's algorithm designed to produce such a subgraph.

- First, we sort all the edges in order of increasing length. (You may assume all edge lengths are distinct.)
- We then construct a subgraph $H = (V, F)$ by considering each edge in order.

- When we come to edge $e = (u, v)$, we add e to the subgraph H if there is currently no u-v path in H. (This is what Kruskal's algorithm would do as well.) On the other hand, if there is a u-v path in H, we let d_{uv} denote the total length of the shortest such path; again, length is with respect to the values $\{\ell_e\}$. We add e to H if $3\ell_e < d_{uv}$.

In other words, we add an edge even when u and v are already in the same connected component, provided that the addition of the edge reduces their shortest-path distance by a sufficient amount.

Let $H = (V, F)$ be the subgraph of G returned by the algorithm.

(a) Prove that for every pair of nodes $u, v \in V$, the length of the shortest u-v path in H is at most 3 times the length of the shortest u-v path in G.

(b) Despite its ability to approximately preserve shortest-path distances, the subgraph H produced by the algorithm cannot be too dense. Let $f(n)$ denote the maximum number of edges that can possibly be produced as the output of this algorithm, over all n-node input graphs with edge lengths. Prove that

$$\lim_{n\to\infty} \frac{f(n)}{n^2} = 0.$$