CS 4820, Summer 2010 Out: July 27. **Due: Friday, July 30, 8:30am**

Homework 6

Reading: Chapter 7.

Problem 1. (15 points) In this problem, we will see a variant of the Ford-Fulkerson algorithm that runs in *strongly polynomial time*, as opposed to weakly polynomial time for the Ford-Fulkerson algorithm. In particular, the running time bound does not depend on actual capacities, only on the number of arcs and vertices in the flow network.

The only difference from the Ford-Fulkerson algorithm is the following: When there is a flow f in an iteration, Ford-Fulkerson algorithm chooses *any* s-t path, say P, in the augmenting graph G_f, and augments the flow f by $\text{bottleneck}_f(P)$ amount along the path P, resulting in new flow f'. It then repeats with new flow f'. In this modification, instead of finding *any* path from s to t in G_f, we find a *shortest* path from s to t in G_f (in terms of *number of arcs*), and augment along this shortest path. Call the new flow f' and repeat with f' as the new flow. Here is the complete description.

```
1 Let f be the current flow. Set f(a) = 0 for all a ∈ A
2 Let G_f be the augmenting graph for flow f.
3 while G_f contains a path from s to t
4     Choose a shortest (simple) path P from s to t in G_f, that is, a path
        that uses the minimum number of arcs in going from s to t.
5     Augment flow bottleneck_f(P) along path P. Call the new flow f'.
6     Set f ← f'.
7     Update G_f
```

In the Ford-Fulkerson algorithm, we used the fact that if all capacities are integers, then every augmentation step (every iteration of the while loop) increases the flow amount by 1 unit at the least, and if the max flow value if C, then the algorithm must terminate after at most C iterations of the while loop.

For our modified algorithm, we use a different fact. We derive its polynomial running time by a sequence of facts.

(a) Let $H = (U, B)$ be an arbitrary directed graph, and let u and v be two distinguished vertices in H. Let d be the length of any shortest path from u to v in H (that is, any path from u to v in H uses at least d arcs, and there is at least one path that uses exactly d arcs). Let $b = (x, y) \in B$ be an arc that lies on at least one shortest path from u to v in H, and let $b^{-1} = (y, x)$. Let $H' = (U, B \cup \{(y, x)\})$, that is, H' is the graph H with arc b^{-1} added to it.

Prove that in H', the shortest path from u to v is at least d, and no length d path from u to v in H' uses arc $b^{-1} = (y, x)$. That is, addition of the new arc b^{-1} is useless as far as finding length d paths from u to v is concerned.

(b) Using part (a), prove that the length of the shortest path from s to t is G_f is non-decreasing throughout the modified Ford-Fulkerson algorithm.

(c) Prove that the length of the shortest path from s to t cannot remain the same in $2m$ consecutive iterations of the while loop. That is, in $2m$ consecutive iterations, the length of the shortest augmenting path must increase by at least 1.

(d) Prove that the modification of Ford-Fulkerson that augments along a shortest path in the graph G_f terminates in $O(mn)$ iterations of the while loop, hence finding the maximum flow in $O(mn)$ iterations.

(e) Analyze the running time of this modified Ford-Fulkerson algorithm.

Problem 2. (10 points) You've been called in to help some network administrators diagnose the extent of a failure in their network. The network is designed to carry traffic from a designated source node s to a

designated target node t, so we will model it as a directed graph $G = (V, A)$, in which the capacity of each arc is 1, and in which each node lies on at least one path from s to t.

Now, when everything is running smoothly in the network, the maximum s-t flow in G has value k. However, the current situation — and the reason you're here — is that an attacker has destroyed some of the arcs in the network, so that there is now no path from s to t using the remaining (surviving) arcs. For reasons that we won't go into here, they believe the attacker has destroyed only k arcs, the minimum number needed to separate s from t (i.e. the size of a minimum s-t cut); and we'll assume they're correct in believing this.

The network administrators are running a monitoring tool on node s, which has the following behavior: if you issue the command ping(v), for a given node v, it will tell you whether there is currently a path from s to v. (So ping(t) reports that no path currently exists; on the other hand, ping(s) always reports a path from s to itself.) Since it's not practical to go out and inspect every arc of the network, they'd like to determine the extent of the failure using this monitoring tool, through judicious use of the ping($\cdot$) command.

So here's the problem you face: give an algorithm that issues a sequence of ping commands to various nodes in the network, and then reports the full set of nodes that are not currently reachable from s. You could do this by pinging every node in the network, of course, but you'd like to do it using many fewer pings (given the assumption that only k arcs have been deleted). In issuing this sequence, your algorithm is allowed to decide which node to ping next based on the outcome of earlier ping operations.

Give an algorithm that accomplishes this task using only $O(k \log n)$ pings.

Problem 3. (5 points) An arc of a flow network is called *critical* if decreasing the capacity of this arc results in a decrease in the maximum flow value. Give an efficient algorithm that finds a critical arc in a network.

Bonus problem: (Warning: it is tricky, not worth much (may be 5–10 points) and no partial credit; basically just for fun if you're bored done with the rest of the problems.)

Consider the algorithm in problem 1, but instead of augmenting along *shortest paths*, you augment along the paths which have the largest residual capacity. So, in every iteration of the while loop, you find a path in G_f which has the highest value of bottleneck$_f(P)$.

(a) Prove that the path with highest value of bottleneck$_f(P)$ can be found using a variant of breadth-first search (or that of Dijkstra's algorithm). Note that Dijkstra algorithm is actually a variant (albeit a clever one) of breadth-first search too!

(b) Show that any flow f in a directed graph with m arcs can be represented as a sum of at most m individual path-flows (m is the number of arcs). Here, a path flow is a flow along a single s-t *simple* path (simple path is a path that does not visit any vertex more than once). That is, path flow is flow represented by a pair (P, x) such that for all arcs $a \in P$, flow on a is x and 0 o.w.

(c) Assume that the capacities are all *integers*. Prove that the algorithm that chooses the path with highest value of bottleneck$_f(P)$ terminates in $O(m \log |F|)$ iterations of the while loop, where m is the number of arcs, and $|F|$ is the value of the maximum flow.

Hint: Try to prove that in first $O(m)$ iterations of the while loop, the flow value increases from 0 to a (small) constant times $|F|$ (like $\frac{1}{e}|F|$ or $\frac{1}{4}|F|$).

Note that we still cannot handle real numbers as capacities with this augmenting path selection rule. The rule in Problem 1 (that of selecting the path with minimum number of arcs in it) works for real capacities.