Homework #10
Introduction to Algorithms/Algorithms 1
600.363/463
Spring 2014

Due on: Tuesday, April 15th, 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 8, 2014

Problem 1 (20 points)

Suppose we have a set of m clients, $C = \{c_1, c_2, \ldots, c_m\}$ each of which must access some website. Suppose further that we have a set of n servers, $S = \{s_1, s_2, \ldots, s_n\}$, each of which hosts some set of websites. For each client c_i, there is a set of servers $S_i \subseteq S$, the set of servers that have the webpage that c_i wants to access. For each client c_i, call S_i the *acceptable server set* for client c_i. Each server s_j can serve at most ℓ_j clients, where $\ell_j \in \{0, 1, 2, \ldots\}$ for all $1 \leq j \leq n$. For each server s_j, call ℓ_j the *service limit* of server s_j. Given a set of clients $C = \{c_1, c_2, \ldots, c_m\}$ and a set of servers $S = \{s_1, s_2, \ldots, s_n\}$, and given the acceptable servers for each client and the service limit for each server, we would like to devise a plan whereby we decide which server each client is going to access to get the webpage it wants. That is, we wish to assign, for each client $c_i \in C$, a server $s_j \in S$ that client c_i is going to access. For each $1 \leq j \leq n$, at most ℓ_j clients can access server s_j. We wish to come up with a *valid assignment* of clients to servers– one that respects the servers' service limits– while also ensuring that as many clients as possible access their desired webpages. An assignment is a set of pairs of the form (c_i, s_j). Each such pair (c_i, s_j) denotes that client c_i

accesses server s_j. An assignment M is valid if

(a) No client accesses more than one server (a client only needs to access one server to retrieve the webpage it wants). That is, for all i, there is at most one pair in M of the form (c_i, s_j).

(b) No server serves more clients than its service limit. That is, for all j, there are at most ℓ_j pairs in M of the form (c_i, s_j).

A valid assignment M is *maximum* if it is the largest valid assignment (i.e., $|M|$ is maximized while still being a valid assignment). Note that by definition, a valid assignment M must have $|M| \leq \min\{m, \sum_{j=1}^{n} \ell_j\}$.

Your job is to give an algorithm that takes as input

(i) a set of clients $C = \{c_1, c_2, \ldots, c_m\}$,

(ii) a set of servers $S = \{s_1, s_2, \ldots, s_n\}$,

(iii) a set of acceptable server sets $A = \{S_1, S_2, \ldots, S_m\}$ with $S_i \subseteq S$ for all $1 \leq i \leq m$ and

(iv) a set of service limits $L = \{\ell_1, \ell_2, \ldots, \ell_n\}$,

and produces as output a maximum valid assignment of clients to servers.

Your algorithm should run in $O(m^2 n)$ time. Prove the correctness and runtime of your algorithm. Hint: consider the similarities of this problem to bipartite matching.

Problem 2 (20 points)

In class, we discussed how Ford-Fulkerson can, in some circumstances, take quite a long time to terminate (or, in some cases, doesn't terminate at all), depending on how we choose our augmenting paths.

Part 1 (10 points)

Give an example flow network on 4 nodes with integer capacities on which Ford-Fulkerson must recompute the residual network at least 999 times (i.e., we must find an augmenting path at least 999 times), if we choose the wrong augmenting path. Note: don't overthink this– If you find yourself needing to include a picture in your writeup you're probably making things more complex than is necessary. Give a brief (2 or 3 sentences) explanation as to why Ford-Fulkerson must recompute the augmenting path so many times.

Part 2 (10 points)

The Edmonds-Karp algorithm is an improvement to Ford-Fulkerson that avoids the problem we discussed in Part 1. Instead of choosing an augmenting path arbitrarily, we do the following:

(1) Given the residual network G_f, compute yet another graph H_f, which is identical to G_f except that all of its residual capacities are 1.

(2) Find the shortest path from the source node s to the sink node t in H_f. Call the resulting path P.

(3) Use P as the augmenting path.

Put another way, Edmonds-karp chooses the augmenting path by finding the shortest path from source to sink as G_f as measured by number of edges. What augmenting path would the Edmonds-Karp algorithm choose in your example graph from Part 1 above? How many times would we have to compute an augmenting path before termination if we follow Edmonds-Karp instead of the augmenting path that you used in Part 1?

Part 3 (Optional, 10 extra credit points)

Give an example of a flow network on which Ford-Fulkerson will never terminate for some unfortunate choice of augmenting paths. Show why Ford-Fulkerson will not terminate on this graph using these augmenting paths. You need not give a rigorous proof, but at least a paragraph of explanation is probably in order. Note: the solution to this problem is all too easy to find on the internet. There is no shame in using, for example, the example given on the Wikipedia page for Ford-Fulkerson, but your solution must be in your own words.

Optional exercises

Solve the following problems and exercises from CLRS: 26.2-7, 26.2-10, 26.2-13, 26.3-2.