Problem Set 7 Solutions

This problem set is due **at 11:59pm** on **Thursday, April 9, 2015.**

Exercise 7-1. Read CLRS, Sections 26.1-26.3.

Exercise 7-2. Exercise 26.1-2.

Exercise 7-3. Exercise 26.1-4.

Exercise 7-4. Exercise 26.2-4.

Exercise 7-5. Exercise 26.2.6.

Problem 7-1. Maximum Flow in a Dynamic Network [20 points]

In this problem, you will design an algorithm that takes the following inputs:

- A flow network $F = (G, c)$, where $G = (V, E)$ is a graph with source vertex s and target vertex t, and c is a capacity function mapping each directed edge of G to a nonnegative integer;
- A maximum flow f for F; and
- A triple (u, v, r), where u and v are vertices of G and r is a nonnegative integer $\neq c(u, v)$.

The algorithm should produce a maximum flow for flow network $F' = (G, c')$, where c' is identical to c except that $c'(u, v) = r$. The algorithm should run in time $O(k \cdot (V + E))$, where $|c(u, v) - r| = k$. The algorithm should behave differently depending on whether $r > c(u, v)$ or $r < c(u, v)$.

(a) [4 points] Start by proving the following basic, general results about flow networks:

1. Increasing the capacity of a single edge (u, v) by 1 can result in an increase of at most 1 in the max flow.
2. Increasing the capacity of a single edge (u, v) by a positive integer k can result in an increase of at most k in the max flow.
3. Decreasing the capacity of a single edge (u, v) by 1 can result in a decrease of at most 1 in the max flow.
4. Decreasing the capacity of a single edge (u, v) by a positive integer k can result in a decrease of at most k in the max flow.

Solution:

1. If (u, v) is in every min cut, then increasing the capacity of (u, v) by 1 increases the min cut value by 1. If (u, v) is not in every min cut, then increasing the capacity of (u, v) by 1 leaves the min cut value unchanged. Either way, the capacity increases by at most 1. The claim follows from the max-flow-min-cut theorem.
2. Increasing by k is the same as increasing in steps of 1. By part 1, each such step increases the max flow by at most 1. So the total increase is at most k.
3. If (u, v) is in some min cut, then decreasing the capacity of (u, v) decreases the min cut value by 1. If (u, v) is not in any min cut, then decreasing the capacity of (u, v) by 1 leaves the min cut value unchanged. Either way, the capacity decreases by at most 1. The claim follows from the max-flow-min-cut theorem.
4. Decreasing by k is the same as decreasing in steps of 1. By part 4, each such step decreases the max flow by at most 1. So the total decrease is at most k.

(b) [8 points] Suppose that $r > c(u, v)$. Describe your algorithm for this case in detail, prove that it works correctly, and analyze its time complexity (in terms of V, E, and k).

Solution: Then the max flow might increase; by Part (a) 2, it increases by at most k.
Algorithm: Regard the existing flow f as a flow in the new flow network $F' = (G, c')$. Start with the residual network of F' for flow f.
Repeat k times:

1. Look for an augmenting path in the residual network.
2. If you find one, add it to the existing flow, else return.

Analysis: Using DFS to search for an augmenting path, each pass through the loop takes time $O(V + E)$, so the total time is $O(k \cdot (V + E))$.
Proof: As noted, increasing the capacity of a single edge by k can result in an increase of at most k in the value of the max flow. Each time we find an augmenting path, it increases the flow by at least 1. So within at most k tries, we reach the max flow.

(c) [8 points] Suppose that $r < c(u, v)$. Describe your algorithm for this case in detail, prove that it works correctly, and analyze its time complexity.

Solution: Then the max flow might decrease; by Part (a) 4, it decreases by at most k.
Algorithm: The algorithm consists of two phases, first removing some flow to fit the reduced capacity of the new flow network $F' = (G, c')$, and then restoring flow to achieve the new max flow.
In Phase 1, if $f(u, v) \leq r$ then we do not reduce any flows. Otherwise, we reduce the flow on (u, v) one unit at a time, until it reaches r. For each unit, we proceed as follows.

First reduce the flow on (u, v) by 1. Then, using DFS, search backwards from vertex u, following incoming edges with positive (incoming) weight, looking for a simple reverse path to s, t, or v. To see why such a path exists, proceed step by step from u, each time following some incoming edge with positive (incoming) weight. Such an edge must exist unless we have reached one of the three listed vertices, because we started with a flow satisfying the flow conservation property. Subtract 1 from the flows along each of the edges in this path.

If the path we found ended in v, we have already restored the flow conservation condition throughout the network. Otherwise, vertex v still does not satisfy flow conservation. So in this case, we use DFS again to search forwards from v, following outgoing edges with positive (outgoing) weight, looking for a simple path to t or s. Subtract 1 from the flows along each of the edges in this path.

At this point, we have restored the flow conservation condition while reducing the flow on (u, v) by 1. We repeat this a total of $f(u, v) - r$ times, reducing the flow on (u, v) to r and restoring the flow conservation condition.

The result of this reduction is now a valid flow for the new network $F' = (G, c')$; however, it needs not to be maximum.

In Phase 2, we augment it to restore maximality. We proceed as in Part (b), trying k times to augment the flow from s to t.

Analysis: In Phase 1, each reduction of capacity by 1 takes time $O(V+E)$, using DFS for the two searches. Thus, Phase 1 takes time $O(k \cdot (V + E))$. Phase 2 is like Part (b), and so takes time $O(k \cdot (V + E))$.

Proof: As we noted, after Phase 1, we have a valid flow. Moreover, in each reduction, we have reduced the value of the flow by at most 1. Therefore, in all, we have reduced the value of the flow by at most k.

The value of the max flow can be anywhere between the value of the flow after Phase 1 and the value of the original max flow. The difference between these two values is at most k, so k iterations are enough to restore the max.

Problem 7-2. Disjoint Roads [15 points]

A number k of trucking companies, $c_1, \ldots, c_k$, want to use a common road system, which is modeled as a directed graph, for delivering goods from source locations to a common target location. Each trucking company c_i has its own source location, modeled as a vertex s_i in the graph, and the common target location is another vertex t. (All these $k + 1$ vertices are distinct.)

The trucking companies want to share the road system for delivering their goods, but they want to avoid getting in each other's way while driving. Thus, they want to find k edge-disjoint paths in the graph, one connecting each source s_i to the target t. We assume that there is no problem if trucks of different companies pass through a common vertex.

Design an algorithm for the companies to use to determine k such paths, if possible, and otherwise return "impossible".

Solution: Model this as a max flow problem. Add a special source vertex s, with edges to all the individual sources s_i, each with capacity 1. Also associate capacity 1 with every other edge, as shown in the following figure:

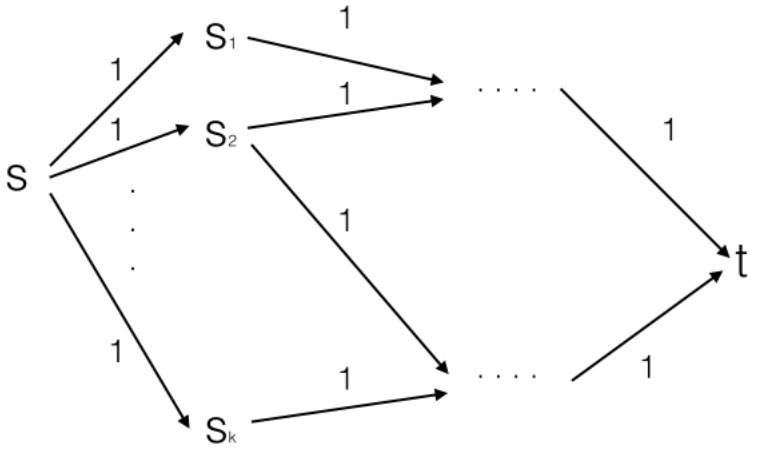

Figure 1: Graph model for problem 7-2.

Proof: Suppose we have any integral flow f on this graph; thus, the flow on every edge is either 0 or 1. From f, we can obtain a collection of disjoint paths to t from those individual sources s_i such that $f(s, s_i) = 1$. We can do this sequentially, one i at a time. Namely, consider any i such that $f(s, s_i) = 1$. Since the flow into s_i is 1, the flow out of s_i is also 1, so identify an edge (s_i, u) such that $f(s_i, u) = 1$. Continue by identifying an edge out of u with $f(u, v) = 1$, and so on, always choosing unused edges. Eventually, this must reach t, so we have discovered a (not necessarily simple) path from s_i to t. Now define a reduced flow by changing all the flows along the discovered path, plus $f(s, s_i)$, to 0. Repeat to identify another path, and continue until we have exhausted all the flow from s to sources s_i and identified paths from all s_i.

Conversely, suppose we have a set of disjoint paths from some subset of the individual sources to s_i. Then we can define a flow on the network by defining $f(u, v) = 1$ for any edge (u, v) that appears in any of the paths, and $f(u, v) = 0$ for other edges.

Thus, we have an immediate correspondence between flows in the network and sets of disjoint paths from subsets of the individual sources. A max flow through this network yields the greatest number of paths, because the definition of a max flow says that it maximizes the total flow out of s (which here corresponds to the number of individual sources that have paths to t).

Algorithm: So, to solve the problem, all we need to do is run a max flow algorithm on the network derived from the truck problem, and interpret the result as a set of paths. If the value of the flow is strictly less than k, we return "impossible".

Analysis: The max flow $|f| < k$. Using the Ford-Fulkerson algorithm, the time complexity is $O(E|f|) = O(Ek)$.

Problem 7-3. Food Truck Orders [15 points]

The Miso Good food truck produces a large variety of different lunch menu items. Unfortunately, they can only produce their foods in limited quantities, so they often run out of popular items, making customers sad.

To minimize sadness, Miso Good is implementing a sophisticated lunch-ordering system. Customers text in their acceptable choices before lunch time. Then they can use an algorithm to preassign lunches to customers. Customers who do not get one of their choices should receive a $\$10$ voucher. Miso Good would like to minimize the number of vouchers they give out.

Give an efficient algorithm for Miso Good to assign lunches to customers. In general, suppose that, on a given day, Miso Good has produced m types of food items $b_1, \ldots, b_m$, and the quantity of each type of food item b_j is exactly q_j. Suppose that n customers $a_1, \ldots, a_n$ text in their preferences, where each customer a_i submits a set A_i of one or more acceptable lunch choices. The algorithm should assign each customer either one of his/her choices or a $\$10$ voucher. It should minimize the number of vouchers.

(*Hint:* Model this as a max flow problem.)

Solution: Model this as a max flow problem. Define a flow network as follows.

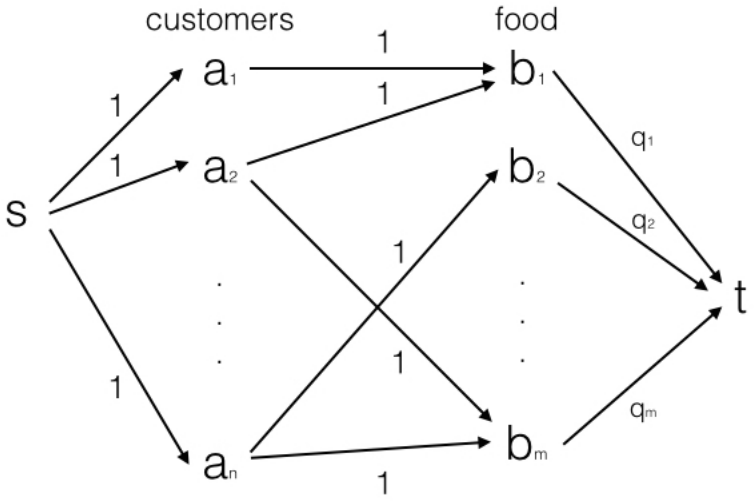

Figure 2: Graph model for problem 7-3, in which a_1 prefers b_1, a_2 prefers b_1 and b_m,... a_n prefers b_2 and b_m.

Include special vertices s and t as usual. Have a "first layer" of n vertices corresponding to the customers, and a "second layer" of m vertices corresponding to the foods. Include an edge from s to each customer, with capacity 1. Include an edge from customer a_i to food item b_j exactly if $j \in A_i$; this will also have capacity 1. Include an edge from food item b_j to t with capacity q_j.

Proof: A flow f on this network yields an assignment of food items to customers that satisfies the customer and food quantity constraints. Specifically, for each customer a_i, if some edge $f(a_i, b_j)$ has flow 1, then assign food item b_j to customer a_i. Note that each customer a_i can get at most one food item, because a_i has incoming flow at most 1, so its outgoing flows must also total at most 1. Since all flows are integral, only one outgoing edge can have positive flow. Also note that each food item b_j cannot be assigned to more than q_j customers, because b_j has outgoing flow at most q_j, so its incoming flows must also total at most q_j. Thus, the food assignment arising from flow f satisfies all the customer and food constraints.

Conversely, any food assignment satisfying all these constraints corresponds directly to a flow through the network: Assign flow 1 to edge (a_i, b_j) exactly if customer a_1 gets assigned food item b_j. Assign flows to the edges from s and to t to achieve flow conservation.

Moreover, a max flow through this network satisfies the maximum number of customers, because the definition of a max flow says that it maximizes the total flow out of s (which corresponds to the number of customers satisfied).

Algorithm: So, all we need to do is run a max flow algorithm on this network, produce an integral max flow f, and interpret it as a food assignment. The maximum number of satisfied customers yields the minimum number of vouchers.

Analysis: The number of edges is $E = O(mn)$. The max flow $|f| \leq n$. Using the Ford-Fulkerson algorithm, the time complexity is $O(E|f|) = O(mn^2)$.

6.046J / 18.410J Design and Analysis of Algorithms
Spring 2015