CS 170 HW 9

Due **2020-03-23, at 10:00 pm**

1 Study Group

List the names and SIDs of the members in your study group. If you have no collaborators, you must explicitly write "none".

2 Flow vs LP

You play a middleman in a market of n suppliers and m purchasers. The i-th supplier can supply up to $s[i]$ products, and the j-th purchaser would like to buy up to $b[j]$ products.
However, due to legislation, supplier i can only sell to a purchaser j if they are situated at most 1000 miles apart. Assume that you're given a list L of all the pairs (i, j) such that supplier i is within 1000 miles of purchaser j. Given $n, m, s[1..n], b[1..m], L$ as input, your job is to compute the maximum number of products that can be sold. The run-time of your algorithm must be polynomial in n and m.
For part (a) and (b), assume the product is divisible, that is, it's OK to sell a fraction of a product.

(a) Show how to solve this problem, using a network flow algorithm as a subroutine. Describe the graph and explain why the output from the network flow algorithm gives a valid solution to this problem.

(b) Formulate this as a linear program. Explain why this correctly solves the problem, and the LP can be solved in polynomial time.

(c) Now let's assume you *cannot* sell a fraction of a product. In other words, the number of products sold by each supplier to each purchaser must be an integer. Which formulation would be better, network flow or linear programming? Explain your answer.

3 Feasible Routing

In this problem, we explore a question called *feasible routing.* Given a directed graph G with edge capacities, there are a collection of supply nodes and a collection of demand nodes. The supply nodes want to ship out flow, while the demand nodes want to receive flow. The question is whether there exists a flow that satisfies all supply and demand.
Formally, we are given a capacitated directed graph, and each node is associated with a *demand value*, d_v. We say that v is a supply node if it has a negative demand value (namely, flow out > flow in), and a demand node, it has a positive demand value (namely, flow in > flow out). A node can be neither demand or supply node, in which case $d_v = 0$. Let $c(u, v)$ be the capacity of the directed edge (u, v). Define a *feasible routing* as a flow that satisfies

- Capacity constraint: for each $(u, v) \in E$, $0 \leq f(u, v) \leq c(u, v)$.
- Supply and demand constraint: for each vertex v, $f^{\text{in}}(v) - f^{\text{out}}(v) = d_v$.

Here, $f^{\text{in}}(v), f^{\text{out}}(v)$ are the sum of incoming flow and outgoing flow at node v.
Note that this is a feasibility problem, and the answer is simply yes or no, whereas the max flow is an optimization problem, where the answer is a number (max flow value).

(a) Let S denote the supply nodes and T the demand nodes. Define the total demand as $\sum_{u \in T} d_u$ and total supply as $\sum_{u \in S} -d_u$. Is there a feasible routing if total demand does not equal total supply? Explain your answer.

(b) Provide a polynomial-time algorithm to determine whether there is a feasible routing, given the graph, edge capacities and node demand values. Analyze its run-time and prove correctness.
Hint: reduce to max flow.

4 Applications of Max-Flow Min-Cut

Review the statement of max-flow min-cut theorem and prove the following two statements.

(a) Let $G = (L \cup R, E)$ be a unweighted bipartite graph. Then G has a perfect matching if and only if, for every set $X \subseteq L$, X is connected to at least $|X|$ vertices in R. You must prove both directions.
Hint: Use the max-flow min-cut theorem.

(b) Let G be an unweighted directed graph and $s, t \in V$ be two distinct vertices. Then the maximum number of edge-disjoint s-t paths equals the minimum number of edges whose removal disconnects t from s (*i.e.*, no directed path from s to t after the removal).
Hint: show how to decompose a flow of value k into k disjoint paths, and how to transform any set of k edge-disjoint paths into a flow of value k.

5 Faster Maximum Flow

In the class, we see that the Ford-Fulkerson algorithm computes the maximum flow in $O(mF)$ time, where F is the max flow value. The run-time can be very high for large F. In this question, we explore a polynomial-time algorithm whose run-time does not the depend on the flow value. Recall that Ford-Fulkerson provides a general recipe of designing max flow algorithm, based on the idea of *augmenting path* in residual graph:

Algorithm 1: Ford-Fulkerson

while *there exists an augmenting path in G_f* **do**
- Find an arbitrary augmenting path P from s to t;
- Augment flow f along P;
- Update G_f

This problem asks you to consider a specific implementation of the algorithm above, where each iteration we find the augmenting path with the smallest number of edges and augment along it.

Algorithm 2: Fast Max Flow

while *there exists an augmenting path in G_f* **do**
  Find the augmenting path P from s to t with the smallest number of edges;
  Augment flow f along P;
  Update G_f

We first show that each iteration can be implemented efficiently. Then we analyze the number of iterations required to terminate. Throughout, we define the shortest path from u to v as the path from u to v with the smallest number of edges (instead of to the smallest sum of edge capacities). Consequently, we define distance $d(u, v)$ from u to v as the number of edges in the shortest path from u to v.

(a) Show that given G_f, the augmenting path P from s to t with the smallest number of edges can be found in $O(m + n)$ time.
Hint: Use the most basic graph algorithm you know.

(b) Show that the distance from s to v in G_f never decreases throughout the algorithm, for any v (including t).
Hint: Consider what augmentation does to the "forward edges" going from u to u', where $d(s, u') = d(s, u) + 1$.

(c) Show that with every m flow augmentations, the distance from s to t must increase by (at least) 1.
Hint: On an augmenting path in the residual graph, call an edge bottleneck if its capacity is the smallest. Observe that the bottleneck edges are removed by each flow augmentation.

(d) Conclude by proving that the total number of flow augmentations this algorithm performs is at most $O(mn)$. Analyze the total run-time of the algorithm.
Hint: You may observe that the distance from s to t can increase at most $O(n)$ times throughout the algorithm. If you use this fact, explain why it holds.