Problem set 1

Electronic submission to Gradescope due **11:59pm Thursday 1/31**. Form a group of 2-3 students — that is, submit one homework with all of your names.

[You may discuss these problems with your classmates, but please do not look for answers to these problems on the Internet. **Your submission must be the original work of you and your partners, and you must understand everything that is written on your submission.** We strongly suggest that you type up your solutions in LaTeX. A template is provided with the class notes.]

**Problem 1**

This problem explores “path decompositions” of a flow. The input is a flow network (as usual, a directed graph $G = (V,E)$, a source $s$, a sink $t$, and a positive integral capacity $u_e$ for each edge), as well as a flow $f$ in $G$. As always with graphs, $m$ denotes $|E|$ and $n$ denotes $|V|$. 

(a) A flow is acyclic if the subgraph of directed edges with positive flow contains no directed cycles. Prove that for every flow $f$, there is an acyclic flow with the same value of $f$. (In particular, this implies that some maximum flow is acyclic.)

(b) A path flow assigns positive values only to the edges of one simple directed path from $s$ to $t$. Prove that every acyclic flow can be written as the sum of at most $m$ path flows.

(c) Is the Ford-Fulkerson algorithm guaranteed to produce an acyclic maximum flow? [Note: We assume that F-F algorithms always finds an acyclic augmenting path.]

(d) A cycle flow assigns positive values only to the edges of one simple directed cycle. Prove that every flow can be written as the sum of at most $m$ path and cycle flows.

(e) Can you compute the decomposition in (d) in $O(mn)$ time?

**Problem 2**

Consider a directed graph $G = (V,E)$ with source $s$ and sink $t$ for which each edge $e$ has a positive integral capacity $u_e$. Recall from Lecture 5 that a blocking flow in such a network is a flow $\{f_e\}_{e \in E}$ with the property that, for every $s-t$ path $P$ of $G$, there is at least one edge of $P$ such that $f_e = u_e$. For example, our first (broken) greedy algorithm from Lecture #1 terminates with a blocking flow (which, as we saw, is not necessarily a maximum flow).

The termination condition implies that the algorithm can only halt with a maximum flow. It’s not hard to see that every iteration of the main loop increases $d(f)$, the length (i.e., number of hops) of a shortest $s-t$ path in $G_f$, and therefore the algorithm stops after at most $n$ iterations. Its running time is therefore $O(n \cdot BF)$, where BF is the amount of time required to compute

---

1Thanks to Tim Roughgarden for letting us reuse some of the problems from his course.
a blocking flow in the layered graph $L_f$. We know that $BF = O(m^2)$ – our first broken greedy algorithm already proves this as does Edmonds-Karp – but we can do better. Consider the following algorithm, inspired by depth-first search, for computing a blocking flow in $L_f$:

**Dinic’s Algorithm**

- initialize $f_e = 0$ for all $e \in E$
- **while** there is an $s$-$t$ path in the current residual network $G_f$ **do**
  - construct the layered graph $L_f$, by computing the residual graph $G_f$ and running breadth-first search (BFS) in $G_f$ starting from $s$, stopping once the sink $t$ is reached, and retaining only the forward edges\(^1\)
  - compute a blocking flow $g$ in $G_f$
  // augment the flow $f$ using the flow $g$
  - for all edges $(v, w)$ of $G$ for which the corresponding forward edge of $G_f$ carries flow $(g_{vw} > 0)$ **do**
    - increase $f_e$ by $g_e$
  - for all edges $(v, w)$ of $G$ for which the corresponding reverse edge of $G_f$ carries flow $(g_{wv} > 0)$ **do**
    - decrease $f_e$ by $g_e$

And now the analysis:

(a) Prove that the running time of the algorithm, suitably implemented, is $O(mn)$. [Hint: How many times can Retreat be called? How many times can Augment be called? How many times can Advance be called before a call to Retreat or Augment?] Recall that a forward edge in BFS goes from layer $i$ to layer $(i + 1)$, for some $i$.

(b) Prove that the algorithm terminates with a blocking flow $g$ in $L_f$. [For example, you could argue by contradiction.]

(c) Suppose that every edge of $L_f$ has capacity 1. Prove that the algorithm above computes a blocking flow in linear (i.e., $O(m)$) time. [Hint: can an edge $(v, w)$ be chosen in two different calls to Advance?]
Problem 3

In this problem we will analyze a different augmenting path-based algorithm for the maximum flow problem. Consider a flow network with integral edge capacities. Suppose we modify the Edmonds-Karp algorithm so that, instead of choosing a shortest augmenting path in the residual network $G_f$, it chooses an augmenting path on which it can push the most flow. (That is, it maximizes the minimum residual capacity of an edge in the path.) For example, in the network in Figure 1, this algorithm would push 3 units of flow on the path $s \to v \to w \to t$ in the first iteration. (And 2 units on $s \to w \to v \to t$ in the second iteration.)

Figure 1: Problem 3. Edges are labeled with their capacities, with flow amounts in parentheses.

(a) Show how to modify Dijkstra’s shortest-path algorithm, without affecting its asymptotic running time, so that it computes an $s-t$ path with the maximum-possible minimum residual edge capacity.

(b) Suppose the current flow $f$ has value $F$ and the maximum flow value in $G$ is $F^*$. Prove that there is an augmenting path in $G_f$ such that every edge has residual capacity at least $(F^* - F)/m$, where $m = |E|$. [Hint: if $\Delta$ is the maximum amount of flow that can be pushed on any $s-t$ path of $G_f$, consider the set of vertices reachable from $s$ along edges in $G_f$ with residual capacity more than $\Delta$. Relate the residual capacity of this $(s,t)$-cut to $F^* - F$.]

(c) Prove that this variant of the Edmonds-Karp algorithm terminates within $O(m \log F^*)$ iterations, where $F^*$ is defined as in the previous problem. [Hint: you might find the inequality $1 - x \leq e^{-x}$ for $x \in [0, 1]$ useful.]

(d) Assume that all edge capacities are integers in $\{1, 2, \ldots, U\}$. Give an upper bound on the running time of your algorithm as a function of $n = |V|$, $m$, and $U$. Is this bound polynomial in the input size?
Problem 4

In this problem we’ll revisit the special case of unit-capacity networks, where every edge has capacity 1 (see also Exercise 4).

(a) Recall the notation $d(f)$ for the length (in hops) of a shortest $s-t$ path in the residual network $G_f$. Suppose $G$ is a unit-capacity network and $f$ is a flow with value $F$. Prove that the maximum flow value is at most $F + \frac{m}{d(f)}$. [Hint: use the layered graph $L_f$ discussed in Problem 2 to identify an $s-t$ cut of the residual graph that has small residual capacity. Then argue along the lines of Problem 3(b).

(b) Explain how to compute a maximum flow in a unit-capacity network in $O(m^{3/2})$ time. [Hints: use Dinic’s algorithm and Problem 2(c). Also, in light of part (a) of this problem, consider the question: if you know that the value of the current flow $f$ is only $c$ less than the maximum flow value in $G$, then what’s a crude upper bound on the number of additional blocking flows required before you’re sure to terminate with a maximum flow?]

Problem 5

This problem explores the notion of liquidity in credit networks for specific graphs. We are given a complete directed graph $G = (V, E)$. There is a weight of $w(x, y)$ for every edge $x \rightarrow y$ in $E$ — it is assumed that the weights are integral. Let $c(x, y) = w(x, y) + w(y, x)$ denote the amount of combined trust (i.e. capacity) between $x$ and $y$.

As a convention, if there is no trust between two nodes, we will say $c(x, y) = 0$.

Recall that a transaction from $x$ to $y$ means finding a directed path from $x$ to $y$ and reversing the path (equivalently, for every edge $(u, v)$ in the path, decreasing $w(u, v)$ by 1 and increasing $w(v, u)$ by 1).

For the following problems, a configuration of a credit network $G = (V, E)$ with fixed capacities $c(x, y)$ for all $(x, y) \in E$ will be a nonnegative value of $w(x, y)$ for each $(x, y) \in E$ such that $w(x, y) + w(y, x) = c(x, y)$.

(a) Let $C_1$ and $C_2$ be two configurations. We will say that the two are cycle-equivalent if and only if $C_2$ can be reached from $C_1$ by performing a sequence of “transactions” along cycles (that is, routes that start and end at the same vertex). Show that this is a well-defined equivalence relation.

(b) Let $C_1$ and $C_2$ be cycle-equivalent configurations. Show that there exists a directed path from a vertex $x$ to a vertex $y$ in $C_1$ if and only if there exists such a path in $C_2$.

(c) Let $C_1$ and $C_2$ be cycle-equivalent configurations, and let $p_1$ and $p_2$ be routes between two vertices $x$ and $y$ (where there exists capacity to route flow along $p_1$ in $C_1$ and along $p_2$ in $C_2$). Let $C_1'$ and $C_2'$ be the configurations arrived at by routing flow along $p_1$ and $p_2$ starting at $C_1$ and $C_2$ respectively. Show that $C_1'$ and $C_2'$ are cycle-equivalent.

The above together show that the set of all configurations can be organized into cycle-equivalent classes, and that transactions are a well-defined operator on these classes. That is to say, we can define an operator on cycle equivalent classes that, given a cycle-equivalent class $C$ and a transaction
from $x$ to $y$, picks a representative configuration of $C$, finds a path from $x$ to $y$ if it exists in the configuration, and routes flow along that path. We have showed that this operator always results in the same cycle-equivalence class $C'$, regardless of the initial choice of representative or choice of route.

The next section deals with what are known as Markov chains. A Markov chain is a model of a process that evolves over time. A Markov chain is built on a set of states $X$. At time $t = 0$, a Markov chain starts in some start state $x_0 \in X$. As time increases from $t$ to $t+1$, the Markov chain moves from state $x_t \in X$ to $x_{t+1} \in X$. The choice of which state to advance into at each timestep is chosen randomly, and depends only on the current state of the chain. Formally, for every pair of states $x, y$, we have an associated quantity $P(x, y)$, which we call the chance that a Markov chain at state $x$ will advance to state $y$ at the next timestep. Of course, for any $x$, $\Sigma_y P(x, y) = 1$.

Consider the following Markov chain. Let $\lambda_{xy} \geq 0$ be a real number for every pair $(x, y) \in V \times V$. Let $S$ be the set of all cycle-equivalence classes of configurations. At each timestep $t$, pick source vertex $x$ and a destination vertex $y$ proportional to $\lambda_{xy}$, and execute a transaction on the current cycle-equivalence class from $x$ to $y$. If the transaction cannot be executed, remain at the same state.

For future problems, assume that transaction rates are symmetric, in that $\lambda_{xy} = \lambda_{yx}$ for all pairs of states.

(d) Let $T$ be the set of all states (cycle-equivalent classes of configurations in the credit network) in the Markov chain with a nonzero probability of being reached if the Markov chain starts at the configuration where $w(x, y)$ is either $c(x, y)$ or 0. Show that if $\lambda_{xy} = \lambda_{yx}$, then the stationary distribution of the Markov chain is uniform over all of $T$ (that is to say, the limit as time goes to infinity of the probability that the Markov chain is in a particular state is constant across all states).

Assume all cycle-equivalence classes are reachable in the Markov chain.

Hint: Assume that the probability at each state is the same, and verify that the total incoming probability is equal to the outgoing probability.

Call the following quantity the liquidity from $x$ to $y$.

$$\frac{\text{#(equivalence classes in which a transaction can happen from } x \text{ to } y)}{\text{#(equivalence classes)}}$$

Observe that the Markov chain models a situation where individuals are transacting with each other at random, with some assumptions on the transaction patterns. Since the stationary distribution of the Markov chain is uniform, the probability that an individual in this model will see their transaction fail is exactly the liquidity quantity defined above.

(e) Suppose that a graph $G$ is tree where $c(x, y) = 1$ for all $x$ and $y$ that are neighbors in the tree. What is the liquidity between two vertices $x$ and $y$, where the length of the (unique) simple path from $x$ to $y$ is exactly $d$?

(f) Suppose that a graph $G$ is a cycle on $n$ vertices (i.e. $c(x_i, x_{i+1}) = 1$ for $i$ from 1 to $n-1$, $c(x_n, x_1) = 1$, and $c(x, y) = 0$ for all other pairs of vertices). What is the liquidity from $x_1$ to $x_j$, for $1 < j \leq n/2$?
Problem 6

A doubly stochastic matrix $G$ of size $n$ by $n$ is any matrix where every entry is non-negative, and the sum of entries in every row and every column is equal to one. One can also think of $G$ as a complete bipartite graph, where the left hand side vertices are the row indices of $G$, and the right hand side vertices are the column indices of $G$. In this problem, we will show that $G$ can be decomposed into a convex combination\(^2\) of permutation matrices.\(^3\)

(a) (Warm up - not graded) Convince yourself that a perfect matching can be thought of as a permutation matrix and vice-versa.

(b) Let $e$ be the edge with the smallest non-zero weight $t$ on $G$, show how to find a perfect matching with weight $t$ that contains edge $e$.

(c) Perform Step (2) iteratively, conclude that $G$ can indeed be decomposed into a convex combination of perfect matchings. What is the running time of your algorithm?

---

\(^2\)A convex combination is a special kind of linear combination where the coefficients are non-negative and sum to 1.

\(^3\)A permutation matrix is a doubly stochastic matrix where every entry is an integer, i.e. 0 or 1.