Problem set 1

Electronic submission to Gradescope due 11:59pm Thursday 2/2. 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?

(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 4 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 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 $L_f$:

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 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.](image)

(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 $E$. Let $c(x, y) = w(x, y) + w(y, x)$ denote the amount of combined trust (i.e. capacity) between $x$ and $y$.

A transaction from $x$ to $y$ can happen on $G$ if and only if there exists a directed path from $x$ to $y$ with capacity one. Let $\mathcal{S}(G)$ denote the set of transactions that can happen on $G$. We call two graphs $G_1$ and $G_2$ equivalent, if and only if $\mathcal{S}(G_1) = \mathcal{S}(G_2)$. The liquidity from $x$ to $y$ is defined as:

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

(check out the course material for more details, such as how a transaction changes the edge capacities of the network)

(a) Consider a cycle with vertices $V = \{x_1, x_2, \ldots, x_n\}$ where $c(x_i, x_{i+1}) = 1$, for $i = 1, \ldots, n$ $(x_{n+1} = x_1)$. What is the liquidity between $x_1$ and $x_j$, for $1 < j \leq n/2$?

(b) (Bonus) What if $c(x_i, x_{i+1}) = k$, for $i = 1, \ldots, n$, where $k$ is some fixed integer larger than 1?

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.\(^3\)

\(^2\)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.
(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?