Flows

1. **Show that**, given an algorithm for finding a maximum set edge-disjoint paths in any directed st-graph, you can use it to find a maximum set of vertex-disjoint directed st-paths in any directed graph.

**Show that there is also a way to do this the other way around.**

Let $G$ be the graph in which to find a maximum set of vertex-disjoint paths. Define a new graph $G'$ by splitting up every $v \neq s, t$ into two vertices $v^{\text{in}}, v^{\text{out}}$, with a single edge between them. More precisely:

$$V(G') = \{s^{\text{out}} = s, t^{\text{in}} = t\} \cup \{v^{\text{in}}, v^{\text{out}} : v \in V(G) \setminus \{s, t\}\},$$

$$E(G') = \{u^{\text{out}}v^{\text{in}} : uv \in E(G)\} \cup \{v^{\text{in}}v^{\text{out}} : v \in V(G) \setminus \{s, t\}\}.$$

Then paths in $G'$ correspond to paths in $G$ (just contract the split-up vertices), and vertex-disjoint paths in $G'$ corresponds to edge-disjoint paths in $G$. Indeed, edge-disjoint paths in $G'$ cannot use an edge $v^{\text{in}}v^{\text{out}}$ twice, which means that the corresponding paths in $G$ cannot use the vertex $v$ twice.

Let $G'$ be a graph in which to find a maximum set of edge-disjoint paths. We use a similar construction to above, but a bit more complicated.

For a vertex $v \in G'$ with $k$ incoming vertices and $l$ outgoing vertices, replace it by $k$ vertices $v^{\text{in}}_i$ and $l$ vertices $v^{\text{out}}_j$, and add all edges $v^{\text{in}}_i v^{\text{out}}_j$. Whenever there is an edge $uv \in E(G')$, put an edge $u^{\text{out}}_j v^{\text{in}}_i$, in such a way that every $u^{\text{out}}_j$ has out-degree 1, and every $v^{\text{in}}_i$ has in-degree 1.

Given any path in $G$, you get a path in $G'$ again by collapsing the split-up vertices. And two paths $P, Q$ in $G$ are vertex-disjoint if and only if the corresponding paths $P', Q'$ in $G'$ are edge-disjoint: If $P'$ and $Q'$ have a common edge $uv$, then $P$ and $Q$ in $G$ have a common edge $u^{\text{out}}_j v^{\text{in}}_i$, so also a common vertex; if $P$ and $Q$ have a common vertex, say $v^{\text{in}}_i$, then since this vertex has in-degree 1, they also share a common edge, hence so do $P'$ and $Q'$. 
2. Give an integer program whose optimal solutions correspond to maximum sets of vertex-
disjoint $st$-paths, in a given directed graph $G$ with vertices $s$, $t$.

Give the dual of the relaxation of this program. What objects in the graph do integral
dual optimal solutions correspond to?

The form in the notes with path-variables won’t work here, because we have a condition
at every vertex, that only one path may pass through it. But in the form with edge-
variables we can add a (second) constraint per vertex. Fortunately, this automatically
forces the $x_e \leq 1$ condition, so we can drop that.

**LP for Vertex-Disjoint Paths**

maximize $\sum_{sw \in \delta^{out}(s)} x_{sw}$ with $x \geq 0$, $x \in \mathbb{Z}^{|E|}$,

\[
\sum_{e \in \delta^{in}(v)} x_e - \sum_{e \in \delta^{out}(v)} x_e = 0 \quad \text{for } v \in V \setminus \{s, t\},
\]

\[
\sum_{e \in \delta^{in}(v)} x_e \leq 1 \quad \text{for } v \in V.
\]

The dual comes out similar to for the flow problem, except that both the $y$ and $z$ are
indexed by the vertices.

**Dual of relaxation**

minimize $\sum_{v \in V} z_v$, with $y \in \mathbb{R}^{|V|}$, $z \geq 0$,

\[
y_v - y_u + z_v \geq 0 \quad \text{for } uv \in E, u \neq s, v \neq t,
\]

\[
y_w + z_w \geq 1 \quad \text{for } sw \in \delta^{out}(s),
\]

\[-y_w + z_t \geq 0 \quad \text{for } wt \in \delta^{in}(t).
\]

Again, we can get a simpler version by putting $y_s = 1$, $y_t = 0$.

**Dual of relaxation**

minimize $\sum_{v \in V} z_v$, with $y \in \mathbb{R}^{|V|}$, $z \geq 0$,

\[
y_v - y_u + z_v \geq 0 \quad \text{for } uv \in E,
\]

\[
y_s = 1, y_t = 0.
\]

An integral optimal solutions corresponds to a vertex cut: a minimum set of vertices
such that if you remove them, there is no longer any $st$-path. The $y_v = 1$ correspond
to vertices on the $s$-side of the cut, $y_v = 0$ ones are on the $t$-side, the $z_v = 1$ correspond
to the vertices of the cut, and the $z_v = 0$ to the others.
3. Use flows to give an algorithm for the binary assignment problem: Given a bipartite graph \( G \) with \( c : E(G) \to \mathbb{Z}_{\geq 0} \) and \( d : V(G) \to \mathbb{Z}_{\geq 0} \), find a maximum assignment, i.e. a \( \varphi : E(G) \to \mathbb{Z}_{\geq 0} \) such that for all edges \( e \) we have \( \varphi(e) \leq c(e) \) and for all vertices \( v \) we have \( \sum_{e \in \delta(v)} \varphi(e) \leq d(v) \).

Define a network by directing the edges of \( G \) from one partite set, call it \( A \), to the other, \( B \). Then add an \( s \) and a \( t \), with edge from \( s \) to all vertices in \( A \), and from all vertices of \( B \) to \( t \). Give the edges in \( G \) the capacities \( c \), and give an edge \( sa \) the capacity \( d(a) \), an edge \( bt \) the capacity \( d(b) \).

A flow \( \varphi \) in this network corresponds exactly to an assignment.

4. A path flow \( g \) is a flow such that \( g(e) > 0 \) only on the edges of one directed \( st \)-path. A cycle flow \( h \) is a flow such that \( h(e) > 0 \) only on the edges of one directed cycle.

Prove that any flow \( f \) can be decomposed into path flows and cycle flows, i.e. there is a set \( P \) of path flows and a set \( C \) of cycle flows, with \( |P| + |C| \leq |E(G)| \), such that

\[
f(e) = \sum_{g \in P} g(e) + \sum_{h \in C} h(e) \quad \forall e \in E(G).
\]

Remove all edges with \( f(e) = 0 \). Find any \( st \)-path in the remainder, let \( \beta \) be the minimum \( f(e) \) on this path, and let \( g_1 \) be the flow with value \( \beta \) along this path, 0 everywhere else. Set \( f := f - g \), and repeat the above. Every time, at least one edge of the path is removed (the one where \( f(e) \) was \( \beta \)), so eventually there are no more paths. The \( g_i \) will be the path flows in \( P \).

The remaining \( f \) still satisfies the flow constraint. Take any edge \( e = uv \) where \( f(e) > 0 \), and trace a path from there. This has to end up back at \( u \), so gives a cycle. Again take the minimum \( f(e) \) along that cycle, which gives a cycle flow \( h_1 \). Remove \( h_1 \) and repeat, until there is no edge with \( f(e) > 0 \). Then the \( h_j \) are the cycle flows in \( C \).

The \( g_i \) and \( h_j \) clearly decompose \( f \), and since each time at least one edge was removed, then total number is \( \leq |E(G)| \).

5. Use the first graph below to show that if the augmenting path algorithm chooses the path \( Q \) arbitrarily, then its running time is not polynomial.

Use the second graph below to show that if there are irrational capacities (here \( \phi = (\sqrt{5} - 1)/2 \), so \( \phi^2 = \phi - 1 \)), then the same algorithm may not terminate. Also show that it may not even converge to the right flow value.

If the algorithm keeps choosing an augmenting path involving the 1-edge, we will have \( \alpha = 1 \) every time, and it will take \( 2K \) iterations until the maximum flow is found. So it can arbitrarily many steps for a constant number of vertices, which means that the algorithm is not polynomial.

The second is quite tedious to write out, so I will cheat and refer to someone else’s writeup, which has nicer pictures anyway. On the course website, go to the notes by Jeff Erickson, and there go to lecture 22, second page.