(40) **Problem 1.** We consider the problem of removing negative edges from a graph $G$ with weights $w(i, j)$, that has negative edges but no negative cycles.

**a)** One simple way to remove negative arcs is to simply add a large constant $C$ to each edge weight, so we get new weights $w'(i, j) = w(i, j) + C$. Give an example to show that this transformation can change what the shortest path is (that is, the sequence of vertices, since of course it changes the actual cost of a path).

We now explore a better approach. Suppose we assign a value $h(v)$ to each vertex in the graph (we will explore what these values should be below). We then create new edge weights $w'(i, j) = w(i, j) + h(i) - h(j)$. for each edge $(i, j)$ in $G$ and call this new graph $G'$.

**b)** Show that for a pair of vertices $u, v$ the shortest path from $u$ to $v$ in $G$ and $G'$ are the same (that is, the sequence of vertices, not the sum of the edge weights).

**c)** We now consider how to compute $h(v)$ values so the $w'(i, j) = w(i, j) + h(i) - h(j)$ are all non-negative. We add a dummy vertex $D$ with an arc of weight zero to every vertex in $G$. We then compute the shortest path from $D$ to each vertex in $G$ (using Bellman-Ford) and let $h(v)$ be this shortest distance from $D$ to $v$.

Prove that $w'(i, j) = w(i, j) + h(i) - h(j) \geq 0$ for each edge $(i, j)$ in $G$.

Note that the results of b and c together show that we can find shortest paths in $G'$ and they will be the same as those in $G$ (and we can now use Dijkstra instead of Bellman-Ford). This isn’t of much use if we want to do this once, but it can help if we want to run multiple single source shortest path computations on $G$ (e.g. to solve all-pairs shortest paths, or the setting below).

We apply this result to a (min-cost) flow network $G$ with costs $w(i, j) \geq 0$.

**d)** Recall that we argued that if we have costs $w(i, j)$ on arcs (along with their normal capacities $c(i, j)$ we could find a minimum-cost flow by repeatedly finding the cheapest augmenting path in $G_f$ where the cost of a forward edge $(i, j)$ in $G_f$ is $w(i, j)$ and for a backward edge $(i, j)$ it is $-w(i, j)$. Since this graph had negative edges, we had to use Bellman-Ford to find shortest A-paths. We now show how to instead maintain a graph like $G'$ that has no negative edges, so we can repeatedly find the cheapest A-path using Dijkstra.

Suppose we have a current flow $f$ such that $G_f$ has no negative cycles. We can thus find values $h(v)$ which is the cheapest path from $s$ to $v$ in $G_f$. We can then transform the edge weights as in part c to remove all negative edges and get $G'_f$.

i) Consider a cheapest A-path $P$ in $G'_f$. Show that all edges on $P$ have weight zero in $G'_f$ (that is, if $i, j$ is on $P$, then $w'(i, j) = 0$).

ii) Conclude from i) that if we augment along $P$ to get new flow $\hat{f}$ and then construct $G'_f$ using the edge weights $w'(i, j)$ from $G'_f$, then $G'_f$ will have no negative edges.
e) Show that if we have a flow \( f \) such that there are no negative cycles in \( G_f \), then when we augment along the cheapest path in \( G_f \) we will create a new flow \( f' \) such that there are no negative cycles in \( G_{f'} \).

Notes: this result further shows that the strategy of repeatedly augmenting along the cheapest A-path will result in a min cost flow. Also, we can (with a bit of care) maintain a residual graph with no negative edges and thus can use Dijkstra’s algorithm to find each A-path.

(20) Problem 2. Consider the directed Hamilton Path problem (DHP) described on page 480.

a) Show that DHP \( \leq_p \) s, t Hamilton Path
where s, t Hamilton Path takes as input a directed graph \( G=(V,E) \) and two designated vertices in \( V \), and returns yes when there is a Hamilton path starting at \( s \) and ending at \( t \).

b) Consider the problem of finding a shortest simple path in a directed graph (with negative cycles) from \( s \) to \( t \) (thus you are given a directed graph \( G \), edge weights, possibly negative, and two designated vertices). Show that this problem is NP-hard (hint: Use your result of part a)).

(15) Problem 3. Suppose we are given a Traveling Sales Man (TSP) problem where cities are points in the plane and distances are actual Euclidean distances (Call this the Euclidean TSP).

Show that the proof of theorem 8.18 p. 479 in the text does not prove that the Euclidean TSP is NP-hard (this problem is in fact still NP-hard, but you are not asked to prove this). The TSP problem is defined on page 474).

(25) Problem 4. The yes/no clique problem is: given an undirected graph \( G=(V,E) \) and a target integer \( k \), is there a clique of size \( k \)? A clique is a set of vertices \( C \) in \( V \) such that each pair of vertices \( (u,v) \) in \( C \), is also an edge in \( E \) (thus every pair of vertices in a clique are connected by an edge).

a) Show that the yes/no clique problem is in NP (note, this problem is NP-C but you are NOT being asked to prove that).

b) Show that you can use a program which solves the yes/no clique problem to actually find a clique of size \( k \) (when one exists). You should find the clique using a polynomial number of calls to the yes/no clique routine, plus polynomial additional work. Thus you are showing that the problem of finding a clique of a given size is polynomially reducible to yes/no clique.

c) Give a polynomial-time algorithm for yes/no clique— when \( k < c \) for a constant \( c \). Would your algorithm still run in polynomial time if we restrict \( k \) so \( k < \log n \), with \( n \) the number of vertices in the graph?