Problem 1
Given is a weighted undirected graph \( G = (V, E) \) with positive weights and a subset of its edges \( F \subseteq E \). An \( F \)-containing spanning tree of \( G \) is a spanning tree that contains all edges in \( F \) (there might be other edges as well). Design an \( O(n^2) \) or \( O(m \log n) \) algorithm that finds the cost of the minimum-cost \( F \)-containing spanning tree of \( G \).

Problem 2
Let \( G = (V, E, w) \) be a positively weighted directed graph. Give an \( O(n^3) \) or an \( O(nm \log n) \) algorithm that finds the minimum distance from \( u \) to \( v \) for every pair of vertices \( u, v \in V \). Moreover, output the number of distinct paths from \( u \) to \( v \) of length equal to the shortest distance.

Problem 3
The Edmonds-Karp algorithm refines the idea of Ford and Fulkerson in the following way: in every iteration, the algorithm chooses the augmenting path that uses the smallest number of edges (if there are more such paths, it chooses one arbitrarily). Find a graph for which in some iteration the Edmonds-Karp algorithm has to choose a path that uses a backward edge. Run the algorithm on your graph – more precisely, for every iteration (a) draw the residual graph, (b) show the augmenting path taken by the algorithm, and (c) the flow after adding the augmenting path.

Do not make your graph overly complicated.

Problem 4
This problem is about reducing one problem to another problem. Suppose we have two problems \( P \) and \( Q \). And suppose we have an algorithm \( A_Q \) that solves problem \( Q \). Can we then use the algorithm \( A_Q \) to solve problem \( P \), with only little extra work? (In this context by “little” we mean that the number of extra steps is polynomially bounded.) If yes, we say that we reduced problem \( P \) to problem \( Q \).

We will work with the following problems:

Problem \( P \) – Hamiltonian path: Given is an unweighted undirected graph \( G \) and two vertices \( s \) and \( t \). Does there exist a path from \( s \) to \( t \) that goes through every vertex exactly once?

Problem \( Q_1 \) – Longest path: Given is an unweighted undirected graph \( G \) and two vertices \( s \) and \( t \). Find the length of the longest path from \( s \) to \( t \). The path can go through every vertex at most once.

Problem \( Q_2 \) – Shortest path with negative weights: Given is a weighted undirected graph \( G \) with arbitrary weights (positive, negative, or zeros) and two vertices \( s \) and \( t \).
Find the length of the shortest path from $s$ to $t$. The path can go through every vertex at most once.

Here are your tasks:

a) Reduce problem $P$ to problem $Q_1$. In particular, for a graph $G$ and vertices $s, t$ for which you want to find a Hamiltonian path, create an input graph $G_1$ and its vertices $s_1$ and $t_1$ that you’ll use as the input for algorithm $A_{Q_1}$. The algorithm will output $\ell$, the length of the longest path from $s_1$ to $t_1$. Use $\ell$ to determine whether $G$ has an $s$-$t$ Hamiltonian path.

- Given $G$, $s$, and $t$, describe how to construct $G_1$, $s_1$, and $t_1$.
- Given $\ell$ for your $G_1$, $s_1$, and $t_1$, describe how to determine whether $G$ has an $s$-$t$ Hamiltonian path.

b) Reduce problem $P$ to problem $Q_2$.

- Given $G$, $s$, and $t$, describe how to construct $G_2$, $s_2$, and $t_2$, the input for algorithm $A_{Q_2}$. Make sure to specify the edge weights for $G_2$.
- Given $\ell$, the length of the shortest $s_2$-$t_2$ path in your $G_2$, describe how to determine whether $G$ has an $s$-$t$ Hamiltonian path.

Note: you cannot modify the algorithms $A_{Q_1}$ or $A_{Q_2}$ – you do not know how they work! Imagine them being a part of a library for which you cannot see the source code. We often refer to such functions as black boxes.