**Problem 1 (30pts).** Given an undirected simple weighted graph \( G = (V, E) \) (having no self-loops or multi-edges) with two designated vertices \( s, t \in V \) and weight \( w_e \) (unnecessarily positive) for any edge \( e \in E \). A path \( p_{s,t} \) is said to be shortest from \( s \) to \( t \) if the aggregate weight of \( p_{s,t} \), defined as \( c(p_{s,t}) = \sum_{e \in p_{s,t}} w_e \), is the smallest among all paths from \( s \) to \( t \).

A second smallest path is a path \( p'_{s,t} \) with \( c(p'_{s,t}) \geq c(p_{s,t}) \) and contains at least one different edge from \( p_{s,t} \), while \( c(p'_{s,t}) \leq c(q) \) for any other \( s-t \) path \( q \).

1. (15pts) Design an algorithm to find such a second smallest \( s-t \) path. Please describe your algorithm and sketch its correctness. Pseudocode is NOT required.
2. (15pts) Two (or more) paths are edge-disjoint if they have no edges in common. The \( K \) shortest Edge-Disjoint Path Problem is to find the shortest, \( 2^{nd} \) shortest, \( \cdots \), \( K^{th} \) shortest \( s-t \) paths which are Edge-Disjoint. Design an algorithm to find these \( K \) paths. Please describe your algorithm and sketch its correctness. Pseudocode is NOT required. (HINT: translate this into a flow problem.)
3. (Bonus 10pts) Two (or more) paths are node-disjoint if they have no common intermediate nodes. Design an algorithm to find the \( K \) shortest Node-Disjoint Paths. Please describe your algorithm and sketch its correctness. Pseudocode is NOT required. (HINT: Convert to \( K \) shortest Edge-Disjoint Path Problem.)

**Solution:** Note that simple graph is a graph with no self-loop or multiple edges. Directed or undirected simple graphs have the same solution for this question.

(1) Use a similar idea of Floyd-Warshall algorithm.

**Step 1:** Use Floyd-Warshall algorithm to find the shortest paths from \( s \) to all \( v \in V \) and those from all \( v \in V \) to \( t \) in \( O(|V|^3) \) time. Denote the shortest path between \( i \) and \( j \) as \( \pi_1(i,j) \). Initialize the second shortest path as \( \pi_2(i,j) \) with \( \text{length}(\pi_2(i,i)) = 0 \) and \( \text{length}(\pi_2(i,j)) = \infty \) for \( i \neq j \).

**Step 2:**

For every edge \((x, y) \in E\)

Let the path \( \text{tmp}(s,t) \) be the concatenation: \( \pi_1(1,x), (x,y), \pi_1(y,t) \)

if \( \text{tmp}(s,t) \) is not shorter than \( \pi_1(s,t) \) and with at least one different edge from \( \pi_1(s,t) \) \&\&

\( \text{tmp}(s,t) \) is shorter than \( \pi_2(s,t) \)

\( \pi_2(s,t) = \text{tmp}(s,t) \)

**Correctness:** all the possible second shortest \( s-t \) paths are searched. If not, assume \( \pi_3(s,t) \) is the solution that we missed, then \( \text{length}(\pi_2(s,t)) \leq \text{length}(\pi_3(s,t)) < \text{length}(\pi_2(s,t)) \), then starting from \( s \), we can find the first edge \((u,v) \) in \( \pi_3(s,t) \) that is not in \( \pi_2(s,t) \), then the concatenation \((s,u)(u,v)(v,t) \) must have been checked in the algorithm, which makes it impossible to get \( \text{length}(\pi_3(s,t)) < \text{length}(\pi_2(s,t)) \).

**Time Complexity:** \( O(|V|^3) \).

(2) We accept two types of solutions according to two different ways of interpreting the question.

First solution, the question is interpreted as find a set of paths \( \{ P_1, P_2, \cdots, P_K \} \) which are edge-disjoint and their total edge weight is minimum. It is then equivalent to a minimal cost flow problem: each edge \( e \) has a unit capacity and cost \( w_e \), we want to find a minimal cost flow for a flow target \( K \). It can be formulated as:
\[
\begin{align*}
\text{minimize} & \quad \sum_{(u,v) \in E} w_{uv} x_{uv} \\
\text{subject to} & \quad \sum_v x_{uv} - \sum_v x_{vu} = \begin{cases} 
K & \text{if } u = s \\
0 & \text{if } u \neq s, t \\
-K & \text{if } i = t 
\end{cases} \\
& \quad 0 \leq x_{uv} \leq 1, \quad \forall (u, v) \in E
\end{align*}
\]

**NOTE:** The optimal solution of the minimal cost flow problem does not show us which is the \( K \)-th shortest disjoint path. So, if we are interested to know which is each one of the \( K \)-th shortest disjoint path, we have to construct a new network, whose set of edges is the set of edges for which the flow is one. To determine the shortest disjoint path, the second disjoint shortest, the third disjoint shortest, and so up to the \( K \)-th shortest disjoint path, we have to execute \( K \) times the following two steps:

- Compute a shortest path in the resulting network;
- Remove its edges (it results a new network).


Second solution is a heuristic algorithm: the problem is interpreted as finding the edge-disjoint up to \( K^{th} \) shortest \( s-t \) paths. This problem can be approached by the Edmonds-Karp algorithm, which keeps finding the shortest augmenting path. The size of the flow equals to the number of edge-disjoint paths from \( s \) to \( t \), among which the first \( K \) edge-disjoint paths with the lowest length can be taken as a solution.

The discovery process of the edge-disjoint paths is to as follows: each edge \((u, v)\) with initial flow \( f(u, v) = 0 \) and capacity \( c(u, v) = 1 \). After finding an augmenting path, all edges with flow value 1 are inverted at the sign of their weight (new weight is \(-w(u, v)\) if \( f(u, v) = 1 \)). Any two augmenting paths \( s-a-u-v-b-t \) and \( s-c-v-u-d-t \) need to be combined to form two disjoint paths \( s-a-u-d-t \) and \( s-c-v-b-t \). Run Bellman-ford algorithm on the resulting network to get these disjoint paths.

(3) The problem is easily transformed into to the K Shortest edge-disjoint path problem. Let us consider some node \( x \), which is allowed to belong to a single path; let us split node \( x \) into two nodes, \( x' \) and \( x'' \), linked by the zero cost edge \((x', x'')\), which has a unit capacity. Change all the edges \((i, x)\) to edges \((i, x')\) and all the edges \((x, j)\) to edges \((x'', j)\). We only need to determine the \( K \) Shortest Edge-Disjoint Paths in the resulting network, to determine the \( K \) Shortest Node-Disjoint Paths in the original one.
**Problem 2 (30pts).** Let $G = (V, E)$ be a connected, undirected graph. For each edge $e \in E$, we have a cost weight $c_e$. The **minimum-spanning-tree problem** is to find an acyclic subset $T \subseteq E$ that connects all of the vertices and whose total weight $c(T) = \sum_{e \in T} c_e$ is minimized.

1. (15pts) Formulate this problem as a linear program. Show the correctness of your formulation.
2. (15pts) Write down the dual of your LP formulation.

**Solution:**

(1) Primal:

\[
\begin{align*}
\text{minimize} & \quad \sum_{(i,j) \in E} c_{ij} x_{ij} \\
\text{subject to} & \quad \sum_{(i,j) \in E: j \in S} x_{ij} \leq |S| - 1, \quad S \subset V \\
& \quad \sum_{(i,j) \in E} x_{ij} = n - 1 \\
& \quad x_{ij} \geq 0, \quad (i,j) \in E
\end{align*}
\]

(2) Dual:

\[
\begin{align*}
\text{maximize} & \quad (n - 1)y_1 - \sum_{S \subseteq V} (|S| - 1) y_s \\
\text{subject to} & \quad y_1 - \sum_{S \subseteq V: j \in S, (i,j) \in E} y_s \leq c_{ij}, \\
& \quad y_1 \geq 0 \\
& \quad y_s \geq 0
\end{align*}
\]
Problem 3(40pts). Given an undirected graph \( G = (V, E) \) where \( V = \{x_1, x_2, \cdots, x_n\} \). A clique is a subgraph \( G_0 \) of \( G \), where \( G_0 = (V_0, E_0) \) with \( V_0 \subseteq V \), \( E_0 \subseteq E \), and for any \( x_i, x_j \in V_0 \) with \( i \neq j \), \((x_i, x_j) \in E_0\). A clique is called \( m \)-clique if the cardinality (number of vertices) of \( V_0 \) is \( m \).

Assume that \( n \gg 9 \) (\( n \) is much larger than 9) and it takes \( O(1) \) time to check whether \((x_i, x_j) \in E\).

1. (10pts) Design an \( O(n^9) \) algorithm to find a 9-clique in \( G \), if such clique exists; answer "no such a clique" if it does not exist. Please describe your algorithm and sketch its correctness. Pseudocode is NOT required.

2. (15pts) Prove that a set of vertices is a 9-clique if and only if it can be partitioned into 3 disjoint 3-cliques such that the union of any two of them forms a 6-clique.

3. (15pts) Show how to find a 9-clique in \( G \) in time \( O(n^\delta) \) for some \( \delta < 9 \), if such a clique exists. Please describe your algorithm and sketch its correctness. Pseudocode is NOT required. (Hint: consider the fast matrix multiplication problem.)

Solution:

(1). Check all \( \binom{n}{9} \) combinations of 9-node. Since each checking takes \( O(1) \), the time complexity is straightforwardly \( O(n^9) \).

(2). The forward direction is straightforward: if a set is 9-clique, then it can be partitioned into 3 disjoint 3-cliques such that the union of any two of them forms a 6-clique.
The reverse direction is also easy to prove. For any two nodes \( u \) and \( v \) in the 9-node set. If \( u, v \) belong to the same 3-clique, then \((u, v) \in E\). If not, the two 3-cliques they belong to, say \( C_1 \) and \( C_2 \) form a 6-clique, so we still have \((u, v) \in E\). Therefore the 9-node set is a clique.

(3).

Step 1: Find the adjacent matrix \( A \) of a transform network \( G' = (V', E') \):

- \( V' = \{3\text{-cliques in } G\} \) (time complexity \( O(n^3) \))
- \( E' = \{\text{pairs of disjoint 3\text{-cliques whose union is a 6\text{-clique}}\} \) (time complexity \( O(|V'|^2) = O(n^6) \))

Step 2: Compute \( A^3 \) using fast matrix multiplication, and check if there are NONZERO elements on the diagonal of \( A^3 \). If no such elements, return "no 9-clique". (Lemma: any nonzero entries on the diagonal of \( A^3 \) correspond to vertices that reside on cycles of length 3, which is then a part of the 9-clique.) The time complexity of Step 2 depends on the efficiency of the fast matrix multiplication algorithm used. Denote it takes \( O(n^w) \) with \( 2 < w < 2.376 \) time to calculate \( M^2 \) for \( n \times n \) matrix \( M \). Then \( A^3 \) take \( O(2|V'|^w) = O(n^{3w}) < O(n^h) \).

Step 3: Let \( s \) be a vertex of \( G' \) such that the diagonal entry of \( A^3 \) corresponding to \( s \) is nonzero. Find such a vertex \( s \), which takes \( O(|V'|) = O(n^3) \) time. Since \( s \) is certainly a part of a 9-clique, find another two vertices of \( G' \) that form a 9-clique with \( s \), which takes \( O(|V'|^2) = O(n^6) \).
The overall time complexity is less than \( O(n^9) \).