Please solve the following 4 problems, some of which have multiple parts.

1. Let $G = (V, E)$ be a directed acyclic graph whose vertices have labels from some fixed alphabet, and let $A[1 .. ℓ]$ be a string over the same alphabet. Any directed path in $G$ has a label, which is a string obtained by concatenating the labels of its vertices.
   
   (a) Describe and analyze an algorithm that either finds a path in $G$ whose label is $A$ or correctly reports that there is no such path.
   
   (b) Describe and analyze an algorithm to find a path in $G$ whose label has minimum edit distance from $A$.

2. Consider a path between two vertices $s$ and $t$ in an undirected edge-weighted graph $G = (V, E)$. The width of this path is the minimum weight of any edge in the path. The bottleneck distance between $s$ and $t$ is the width of the widest path from $s$ to $t$. (If there are no paths from $s$ to $t$, the bottleneck distance is $-∞$; on the other hand, the bottleneck distance from $s$ to itself is $∞$.) See Figure 1.

(a) Prove that the maximum spanning tree of $G$ contains widest paths between every pair of vertices.

(b) Describe an algorithm to solve the following problem in $O(V + E)$ time: Given an undirected weighted graph $G = (V, E)$, two vertices $s$ and $t$, and a weight $W$, is the bottleneck distance between $s$ and $t$ at least $W$?

(c) Suppose $B$ is the bottleneck distance between $s$ and $t$.
   
   i. Prove that deleting any edge with weight less than $B$ does not change the bottleneck distance between $s$ and $t$.
ii. Many graph algorithms use an operation called **edge contraction**. To contract the edge \( uv \), we insert a new node, redirecting any edge incident to \( u \) or \( v \) (except \( uv \)) to this new node, and then delete \( u \) and \( v \). After contraction, there may be multiple parallel edges between the new node and other nodes in the graph; we then (for the sake of this problem) remove all but the **heaviest** edge between any two nodes. See Figure 2.

![Figure 2: Contracting an edge and removing redundant parallel edges.](image)

Prove that contracting any edge with weight greater than \( B \) does not change the bottleneck distance between \( s \) and \( t \).

(d) Describe an algorithm to compute a minimum-bottleneck path between \( s \) and \( t \) in \( O(V + E) \) time. You may assume that given any subset of edges \( E' \subseteq E \), it is possible to contract every edge in \( E' \) in \( O(V + E) \) time total.

[Hint: Start by finding the median-weight edge in \( G \). Use the observations in part (c) to reduce the size of the graph.]

3. Suppose we are given a directed graph \( G = (V, E) \) with edge weights \( w : E \to \mathbb{R} \) and two vertices \( s \) and \( t \). You may assume \( G \) has no negative weight cycles.

(a) Describe and analyze an algorithm to find the shortest path from \( s \) to \( t \) when exactly one edge in \( G \) has negative weight. [Hint: Modify Dijkstra’s algorithm. Or don’t.]

(b) Describe and analyze an algorithm to find the shortest path from \( s \) to \( t \) when exactly \( k \) edges in \( G \) have negative weight. Any \( O(f(k)E \log V) \) time algorithm where \( f \) is a function of \( k \) is worth full credit, but an \( O(kE \log V) \) time algorithm may be faster and easier to analyze than those with worse dependency on \( k \). [Hint: Modify Bellman-Ford so it sometimes calls a variant of Dijkstra’s algorithm in the subgraph of non-negative weight edges.]
4. In this problem, we will discover how you, yes you, can be employed by Wall Street to cause a major economic collapse! The arbitrage business is a money-making scheme that takes advantage of differences in currency exchange. In particular, suppose 1 US dollar buys 120 Japanese yen, 1 yen buys 0.01 euros, and 1 euro buys 1.2 US dollars. Then, a trader starting with $1 can convert their money from dollars to yen, then from yen to euros, and finally from euros back to dollars, ending with $1.44! The cycle of currencies $\rightarrow$ ¥ $\rightarrow$ € $\rightarrow$ $ is called an arbitrage cycle. Of course, finding and exploiting arbitrage cycles before the prices are corrected requires extremely fast algorithms.

Suppose $n$ different currencies are traded in your currency market. You are given a matrix $\text{Exch}[1..n,1..n]$ of exchange rates between every pair of currencies; for each $i$ and $j$, one unit of currency $i$ can be traded for $\text{Exch}[i,j]$ units of currency $j$. (Do not assume that $\text{Exch}[i,j] \cdot \text{Exch}[j,i] = 1$.)

(a) Describe and analyze an algorithm that returns a matrix $\text{MaxAmt}[1..n,1..n]$, where $\text{MaxAmt}[i,j]$ is the maximum amount of currency $j$ that you can obtain by trading, starting with one unit of currency $i$, assuming there are no arbitrage cycles.

[Hint: Reduce to APSP. How can you turn a problem about maximizing a product into one about minimizing a sum?]

(b) Describe and analyze an algorithm to determine whether the given matrix of currency exchange rates creates an arbitrage cycle.