Problem 1  Give an algorithm that determines whether or not a given undirected graph \( G = (V, E) \) contains a cycle. Your algorithm should run in \( O(|V|) \) time, independent of \( |E| \).

Argue correctness, running time.

Problem 2  Let \( G = (V, E) \) be an undirected multigraph (parallel edges are allowed). A bridge is an edge whose removal disconnects \( G \). Prove that an edge \( e \) is a bridge if and only if there is no simple cycle \( C \) of \( G \) which contains \( e \).

Use this together with the DFS tree \( G_\pi = (V, E_\pi) \) and the classification of edges in a DFS tree to give a \( O(|V| + |E|) \) algorithm to determine all the bridges of \( G \). Give pseudocode and prove correctness. Analyze the running time. If you desire, use a data structure to hold \( G_\pi \); in this case describe what this data structure is (i.e., parent pointers only, etc.).

Hint: for an undirected multigraph, only “back” edges exist. Also use this, the discovery times as computed by DFS, and a post-order traversal of the DFS tree to store at each node information that can determine if the edge from \( v \) to its parent is a bridge edge (this can also be done by modifying the DFS pseudocode).

Problem 3  Present a polynomial-time algorithm for the following problem: Given directed graph \( G = (V, E) \), construct another directed graph \( G' = (V, E') \) such that \( |E'| \) is minimized, and for any \( u, v \in V \), there exists a \( u \rightarrow v \) directed path in \( G \) if and only if there exists a \( u \rightarrow v \) directed path in \( G' \). Note: \( E' \) is not required to be a subset of \( E \).

Analyze the running time and argue that your algorithm is correct. See the discussion in the textbook in Chapter 22.5 on the component graph of a directed graph.

Problem 4  Suppose we are given a weighted directed graph \( G = (V, E, c) \) with (possibly negative) costs on the edges, and where every cycle of \( G \) has strictly positive cost, and two nodes \( u, v \in V \). Give an efficient (polynomial) algorithm for computing the number of shortest \( u \rightarrow v \) paths in \( G \). Do not attempt to list all these paths!

Present pseudocode, analyze the running time, and prove correctness. (I’ve seen a variant given during an interview). Hint: Chapter 24.2 may also supply the right idea.