For all problems, when you design an algorithm, present it in high-level pseudocode, give a proof of correctness, and provide justification for its runtime analysis.

Problem 1 (10 Points) - Maximum-Weight Independent Set in a Tree

Let $T = (V, E)$ be a tree, and assign each vertex a real weight $w(v)$, possibly negative. A set $S \subseteq V$ is an independent set in T when it contains no adjacent vertices.[1] Its weight is

$$w(S) = \sum_{v \in S} w(v).$$

The MaxWeightIS problem is defined by:

Input: a tree $T = (V, E)$ with a real weight $w(v)$ for each vertex $v \in V$

Output: The maximum weight $w(S)$ over all independent sets $S \subseteq V$

(i) Give a counterexample to show that the following greedy algorithm does not solve the MaxWeightIS problem: start with the empty set $S = \emptyset$ and repeatedly select a maximum-weight remaining vertex of positive weight, add it to S, and delete that vertex and all its neighbors from T.

(ii) Design an $O(n)$-time dynamic program that solves MaxWeightIS.

Problem 2 (10 Points) - Parity of Shortest Paths

The ShortestPathsParity problem is defined by:

Input: A DAG $G = (V, E)$ with real edge weights $w(e)$ for each $e \in E$,[2] a start vertex $s \in V$.

Output: For each vertex v in V, the shortest length of an s-to-v path in G, and whether the number of shortest-length paths from s to v in G is odd or even.

For an unreachable vertex v, the output length for paths to v is ∞ and there is an even number (namely, zero) of shortest-length paths from s to v.[3]

Design a dynamic programming algorithm that solves the ShortestPathsParity problem. Your algorithm should run in $O(n + m)$ time.

Remark. Note that the number of total shortest paths may not fit into one word in the word-RAM model.

[1] The empty set is an independent set.

[2] Edge weights may be negative.

[3] Note that s is reachable from itself. The empty path is the unique path from s to itself.

Problem 3 (10 Points) - All-Pairs Bottleneck Paths

Let $G = (V, E, w)$ be a directed graph with nonnegative edge weights, i.e. $w : E \to \mathbb{R}_{\geq 0}$. Define the *bottleneck cost* of a nonempty path P by

$$b(P) := \max_{e \in P} w(e).$$

For each ordered pair u, v, define

$$B(u, v) = \min\{b(P) : P \text{ is a directed path from } u \text{ to } v\}$$

to be the minimum bottleneck cost of a path from u to v. For every vertex $u \in V$, $B(u, u) = 0$. And if no u-to-v path exists in G, $B(u, v) = \infty$.

The APBP (All-Pairs Bottleneck Paths) problem is defined by:

Input: A directed graph $G = (V, E)$ with non-negative edge weights $w(e)$ for each $e \in E$.

Output: The value $B(u, v)$ for each $u, v \in V$.

Design a Floyd–Warshall-style dynamic program that solves the APBP problem in time $O(n^3)$.

Problem 4 (10 Points) - Increasing the Capacity of One Edge

The MaxFlowUpdate problem is defined by:

Input: An undirected graph $G = (V, E)$ with integral capacity $c(e)$ for each $e \in E$, a source vertex $s \in V$, a sink vertex $t \in V$, an integral maximum s-t flow f, and an edge $e \in E$.

Output: A maximum flow $\widetilde{f}$ of the graph G' that equals G and has the same capacities except that the capacity of e is increased by $+1$.

(i) Prove the following claim:

The maximum-flow value increases by 1 if and only if e crosses[4] every minimum s-t cut of the original graph.

(ii) Use the above claim to design an algorithm that solves the MaxFlowUpdate problem in time $O(n+m)$.

[4] "Crosses" means that the endpoints of e lie on different sides of the cut.