Please enter your name and uID below.

Name:
uID:

Submission notes

- Due at 11:59 pm on **Wednesday, December 2**.
- Solutions must be typeset using one of the template files. For each problem, your answer must fit in the space provided (e.g. not spill onto the next page) *without* space-saving tricks like font/margin/line spacing changes.
- Upload a PDF version of your completed problem set to Gradescope.
- Teaching staff reserve the right to request original source/tex files during the grading process, so please retain these until an assignment has been returned.
- Please remember that for problem sets, collaboration with other students must be limited to a high-level discussion of solution strategies. If you do collaborate with other students in this way, you must identify the students and describe the nature of the collaboration. You are not allowed to create a group solution, and all work that you hand in must be written in your own words. Do not base your solution on any other written solution, regardless of the source.

1. (Minimum Flow) Suppose instead of capacities, we consider networks where each edge $u \to v$ has non-negative demand $d(u \to v)$. In this setting, an st-flow is *feasible* if and only if $f(u \to v) \geq d(u \to v)$ for every edge $u \to v$.

   a Describe an efficient algorithm to compute a feasible st-flow.

   b Suppose you have access to an efficient `MaxFlow` subroutine which computes maximum flows in networks with edge capacities. Describe an efficient algorithm to compute a minimum flow in a network with edge demands. Your algorithm should call `MaxFlow` exactly once.

2. (APSP) Let $G = (V, E)$ be a directed graph with weighted edges (edge weights may be positive, negative, or zero). You may assume there are no negative cycles.

   a Describe an efficient algorithm which constructs a directed graph $G' = (V \setminus \{v\}, E')$ with weighted edges such that the shortest-path distance between any two vertices $u, w \neq v$ in G' is equal to the shortest-path distance between u and w in G. The algorithm should run in $O(V^2)$ time.

   b Now suppose we have already computed all shortest-path distances in G'. Describe an efficient algorithm to compute the shortest-path distance from v to every other vertex, and from every other vertex to v. The algorithm should run in $O(V^2)$ time.

   c Combine parts (a) and (b) into an all-pairs shortest path algorithm which runs in $O(V^3)$ time.

3. (BoxDepth) Consider the BoxDepth problem: given a set S of n axis aligned rectangles in the plane (that is, their sides are parallel to the x and y axes), is there a subset $X \subseteq S$ of size at least k such that all of the rectangles in X share a common point (that is, there are values x^* and y^* so that the point (x^*, y^*) is contained in every rectangle in X)?

   a Describe a polynomial-time reduction from BoxDepth to Clique. You must include a description in English of how to transform an arbitrary instance of one problem into the other, clearly note the size of the new instance with respect to the size of the original instance, state and argue the runtime of your transformation. Further, be sure to include both directions of the correctness proof.

   b Describe (in English) a polynomial-time algorithm for BoxDepth and analyze its runtime. [*Hint:* $O(n^3)$ *should be easy.*]

   c Why don't these two results imply that P = NP?

4. (MultiCut) Consider the MultiCut problem: given an undirected graph $G = (V, E)$, c pairs of vertices $(s_1, t_1), \ldots, (s_c, t_c)$, and an integer k, the answer to MultiCut is YES if and only if there is a way to remove at most k edges from G such that s_i is not connected to t_i for any $i = 1, \ldots, c$.

   a Prove that MultiCut is in NP. You should describe a certificate of a YES-instance, as well as the algorithm for verifying it (including its runtime).

   b Prove that MultiCut is NP-hard using a reduction from VertexCover. You must include a description in English of how to transform an arbitrary instance of one problem into the other, clearly note the size of the new instance with respect to the size of the original instance, state and argue the runtime of your transformation. Further, be sure to include both directions of the correctness proof.