You may discuss these problems with your classmates, but you must write up your solutions independently, without using common notes or worksheets. You must indicate at the top of your homework who you worked with. Your write up should be clear and concise. You are trying to convince a skeptical reader that your answers are correct. Your homework should be submitted via Autolab (https://autolab.cs.cmu.edu/02713-s13/) as a typeset PDF. A LaTeX tutorial and template are available on the class website if you choose to use that system to typeset.

1. Let \( G = (V, E) \) be a connected, undirected graph with positive edge weights \( d(u, v) \), and suppose \(|E| = \Omega(|V|^2)\). That is, \(|E|\) is perhaps \((|V|^2)/100\). Give an implementation of Dijkstra’s shortest path algorithm that is asymptotically faster on such dense graphs than the heap-based version we saw in class.

2. Suppose \( G = (V, E) \) is a directed, acyclic graph (a DAG) with positive weights \( d(u, v) \) on each edge. Let \( s \) be a vertex of \( G \) with no incoming edges and such that every other node is reachable from \( s \) through some path.
   (a) Give an \( O(|V| + |E|) \)-time algorithm to compute the shortest paths from \( s \) to all other vertices in \( G \). Note that this is faster than Dijkstra’s algorithm in general.
   (b) Give an efficient algorithm to compute the longest paths from \( s \) to all other vertices. (Interestingly, this is a hard problem in general, non-DAG graphs.)

3. Give an \( O(|E|) \)-time algorithm to check whether a given tree \( T \) is a shortest-path tree of an undirected graph \( G = (V, E) \) with positive weights \( d(u, v) \) on each edge. 
   Hint: check whether the edges that are not in \( T \) are correctly excluded.

4. Often we want to compute shortest paths between all pairs of vertices in a graph \( G = (V, E) \). There are several ways to do this. One way to do this is to run Dijkstra’s algorithm using each vertex as the start node. This takes \( O(|V||E| \log |V|) \) time, but doesn’t work if there are negative weights. Here we explore one fix to that suggested by Edmonds and Karp.
   (a) Give an example where Dijkstra’s algorithm fails if there are edges of negative weight even if there are no negative cycles.
   (b) Let \( G = (V, E) \) be a directed graph with possibly-negative weights \( d(u, v) \) on the edges (but no negative cycles). Define a new graph \( G' \) that is created from \( G \) by adding a new vertex \( s \) and edges of 0-weight from \( s \) to every node in \( V \). Let \( \text{dist}_s(v) \) be the shortest path distance from \( s \) to \( v \) in \( G' \) computed via Bellman-Ford. Define new weights on \( G \) as \( d'(u, v) = d(u, v) + \text{dist}_s(u) - \text{dist}_s(v) \). Argue that \( d'(u, v) \) defined in this way is always positive.
   (c) Argue that any path is a shortest path in \( G \) using weights \( d'(u, v) \) if and only if it is a shortest path in \( G \) using weights \( d(u, v) \). (In other words, changing the weights to \( d'(u, v) \) preserves which paths are shortest paths.)
   (d) Describe how to use (b) and (c) to compute all pairwise shortest paths in \( O(|V||E| \log |V|) \)-time.