Problem 1. Recall that the strongly connected components of a graph G can be found by running DFS on G and then DFS on G^T, considering vertices in decreasing order of the finishing times from the first DFS run. It might seem like it would be equivalent to perform the second run of DFS on G instead of G^T, considering vertices in increasing order of the finishing times. But this, in fact, does not work.

Give an example of a graph G where this proposed simpler algorithm (running DFS twice on G) fails. State the finishing times from the first DFS run, the incorrect components that are found by the simpler algorithm, and the correct components.

Problem 2. Describe an algorithm that finds the spanning tree with *maximum* weight, rather than *minimum* weight as shown in class.

Problem 3. Describe an algorithm that finds the spanning tree with minimum *height*, rather than minimum *weight* as shown in class. Note that the height depends on which node is selected as the root.

Problem 4. DFS is not a suitable algorithm to find the maximum height spanning tree for all connected graphs (in fact, no efficient algorithm for this has been found and such an algorithm likely does not exist).

Give an example of a graph that proves that DFS does not work. Give the tree that would be computed by DFS and the actual maximum height spanning tree. Assume that vertices and adjacency lists are sorted alphabetically.

Problem 5. CLR(S) 24.1-4: Modify the Bellman-Ford algorithm so that it sets `v.d` to $-\infty$ for all vertices `v` for which there is a negative-weight cycle on some path from the source to `v`.

Problem 6. Dijkstra's algorithm does not generally work for graphs with negative weights. Consider the following modification: determine the lowest weight w_{min} of any edge in the graph, add the absolute value of that weight $|w_{\min}|$ to all edges' weights, then run Dijkstra, and finally subtract $|w_{\min}|$ again to adjust the weights of the computed shortest paths.

Show that this modification does not work. Give an example of a graph G for which this modified version of Dijkstra's algorithm computes at least one incorrect shortest path and point out a pair of vertices along with the incorrectly computed path and a correct path.

Problem 7. Modify the pseudocode of the Floyd-Warshall algorithm given in class so that it also computes the matrices $\Pi^{(k)}$ for $0 \leq k \leq n$, containing the predecessor information needed to find the actual shortest paths, not just their weights stored in $D^{(k)}$. Give pseudocode for a function `PRINT-PATH(Π, u, v)` that prints a shortest path for any pair of vertices $u, v \in \{1, 2, \ldots, n\}$ given the matrix $\Pi^{(n)}$.

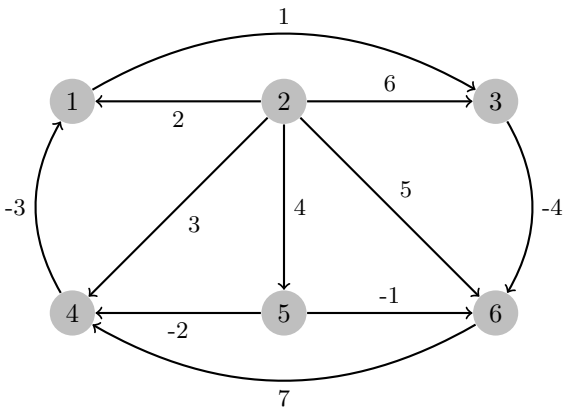
See page 2 for some extra credit problems.

Extra credit

Each of these extra credit problems can make up for any one point that you missed in this assignment or in any previous one. In other words, it counts as much as a regular problem without increasing the total number of problems.

Extra Credit 1. The modified version of Dijkstra's algorithm described in problem 6 does find optimal results for *some* graphs. State a necessary and sufficient condition that a graph with negative weights must satisfy in order for this algorithm to find only correct shortest paths. Argue briefly why your condition is correct.

Extra Credit 2. Compute $D^{(k)}$ for $0 \leq k \leq 6$ for the graph below using the Floyd-Warshall algorithm.



Extra Credit 3. Compute the prefix function π of the KMP algorithm for the pattern `aabaabbaabaaabab`.

Extra Credit 4. Use the Wagner-Fischer algorithm to compute the Levenshtein distance between the words `PERNICIOUS` and `EXPEDITIOUS`. Show the full array of solutions to subproblems.

Extra Credit 5. The Levenshtein distance is based on the three operations of insertion, deletion, and substitution. Expand the recurrence for its computation to also allow for the transposition of two adjacent letters in either word. Briefly explain the new case. Note that you do not need to come up with the solution yourself but can instead try to find the name of this "new" edit distance to find the solution.