COP 4516 Spring 2024 Week 6 Ind: Weighted Graphs (Solution Sketches)

Ant Tunneling
This problem is exactly the minimum spanning tree problem, since it's describing creating a network that connects all of the ant hills at a minimum cost. Either Prim's or Kruskal's algorithm can be used to solve the problem. Since it's possible that the input graph isn't connected, it's important to detect this case. After completing the algorithm, if there are fewer than n – 1 edges in the minimum spanning tree that is built, where n is the number of vertices (ant hills), then no such construction exists and you must output, "I'm a programmer, not a miracle worker!"

Dueling Philosophers
This is a slight twist on the regular topological sort problem. Instead of just answering whether or not there is a valid topological sort, in the case that a topological sort exists, we just determine if there are more than one possible topological sort. Take the usual algorithm and run it to completion. This will distinguish between cases with and without a topological sort. Now, edit as follows: as the algorithm is running, keep track of if there were ever two possible vertices that could have been placed in a particular slot (ie. two vertices with a current in-degree of 0) in the topological sort. This can just be a boolean flag that is initially set to false and toggles to true any time two 0s are noted for in-degrees when the algorithm looks to choose the next vertex of minimum in-degree. At the very end, in the case that a topological sort has been found, we distinguish between outputs of 1 and 2 by simply looking at the value of the boolean flag. If it's false, we return 1, if it's true we return 2.

Come Rain or Shine
We can use either Bellman-Ford or Dijkstra's to determine the shortest path. However, these algorithms don't determine the longest path in a graph. One little trick that works is to negate all of the edge weights in the original graph G, creating a new graph G'. All the distances in G' will be negative (except source to source). If we find the "shortest" distance in this graph, it will correspond to the "most negative" number. If we think about what this means for our original graph G, this corresponds to the "most positive" path length, or longest path length in the negative graph. So one possible solution is as follows: Create G' by negating all of the edges in G. Run Bellman-Ford's on it (since this works with negative edge weights and Dijkstra's doesn't), get the shortest distance to the final destination (a negative number). Then, just negate this to get the longest possible path length in G from source to destination.

Triangular Sums
This is the easiest problem in the set. A definition for W(n) is given to you and you have to calculate W(n) for any input value from 1 to 300. One key observation is that $W(n+1) = W(n) + (n+1)T(n+2)$. Thus, we can simply build up the results, one by one. The triangular numbers themselves have a formula, $T(n) = n(n+1)/2$. (Even if one didn't know that formula, one can create an extra variable to store the sum of the first i integers and add to it.)

Substituting, we have $W(n+1) = W(n) + (n+1)(n+2)(n+3)/2$ and $W(0) = 0$. From there we can just run a single for loop building up these values and storing them in an array. Then, process the input answering each query immediately.