COP 4516 Spring 2025 Week 4 Ind: Weighted Graphs (Solution Sketches)

Spin
The input is a graph with up to 5000 vertices and 10000 edges. We must find a shortest distance (from vertex 0 to vertex n-1, where n is the number of vertices), but there is a catch! Not all routes are allowed. Any route with more than w crosswalks is not allowed for consideration. Normally, Dijkstra's algorithm would be used to find a single shortest distance in a graph, but our added restriction seems to put a damper on this plan.

Ultimately, when tracing through Dijkstra's algorithm, it becomes clear that when arriving at any vertex, **it's also necessary to know how many crosswalks** a particular path took to get there. Arriving at vertex a using 3 crosswalks is simply a different case than arriving at vertex a using 4 crosswalks. Thus, instead of storing a shortest distance for each vertex, **we must store a shortest distance for each ordered pair (v, k) where v is a vertex in the graph and k is the number of crosswalks used to arrive there.**

The effect of doing this is that Dijkstra's is run on a graph with n(w+1) number of vertices, where n is the number of vertices in the graph and w is the maximum number of crosswalks allowed. For these bounds we have 5000 x 21 = 105000 vertices. The effective # of edges is 10000 x 21 = 210000. The run time of Dijkstra's (efficient implementation) is O(ElgV). Thus, by plugging in these maximum values into this run-time, we see that the algorithm is indeed fast enough.

Thus, the key modification to the Dijkstra's code shown in class is to keep a two dimensional Boolean array for "used" vertices (or an equivalent shortest distance array), and when enqueing items into the priority queue, enqueue ordered pairs of vertex number and number of crosswalks. Alternatively, create a single integer, say, n*Crosswalks + v that stores both of these integers in a single integer. Stop looking for paths once ANY path to the class is found.

To solve the small case only, with n ≤ 20, we see that n(w+1) ≤ 420, and for this bound, Floyd Warshall's is fast enough to obtain the shortest distance between the dorm and class. After running Floyd's, take shortest entry of reaching class amongst using any number of crosswalks.

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.

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.