1. [35 Points] Implementing the Floyd-Warshall Algorithm! [Due Friday at midnight] In this problem you will implement the Floyd-Warshall (FW) shortest path algorithm in your favorite programming language.

You can assume the graph is input as follows. The first row is a single positive integer that specifies how many vertices, $n$, there are. The next $n$ rows specify the adjacency matrix for these $n$ vertices. In particular, the first row of weights corresponds to vertex 1, the next to vertex 2, and so on up through vertex $n$. Within a given row, weights correspond (left to right) to vertex $1, 2, \ldots, n$ respectively, and weights are space-separated integers. The edge weight value 10000 represents $\infty$.

Your program should find the shortest path between every pair of vertices and then, for each pair, print out two things:

(a) The distance from $i$ to $j$.

(b) The *shortest path* (list of vertices) from $i$ to $j$ that correspond to this path.

It is a natural mapping from our discussion in class to getting the first item to work (I recommend you start by doing this part). The second item is a bit trickier—think before you code and start early (debugging can be painful). When printing paths, take care to ensure that no self-loops are printed (i.e. do not include in the output any 0-weight $(i, i)$ edges). **Along with your source code, include pseudo-code that describes it at a high level and analyze its run-time.** Your path printing code should *not* be exponential!

I provide two graph definition files, `fw_data1` and `fw_data2`, and corresponding correct results, `fw_data1_results` and `fw_data2_results`, are available at:

- [http://www.cs.hmc.edu/courses/2004/fall/cs140/homework/fw_data1_results](http://www.cs.hmc.edu/courses/2004/fall/cs140/homework/fw_data1_results)

These files serve to document what your program should read as input, how it should format its answers, and provide test-cases for debugging your code.

You should turn in a printout of your source code as well as scripts that show your program’s output on these two graphs (and the aforementioned run-time analysis). I recommend that you deal with input and output using standard I/O. For example, if your program was called `myFW`, run
myFW < fw_data1 > fw_data1_out.

To generate a script, you can then simply: lpr fw_data1_out.

You should take effort to write reasonable, comprehensible code. Badly documented or unmodular code may lose points. Extra credit may be assigned for clean, elegant efforts.