Why are we to use an adjacency matrix to store our graph? The spec doesn't say that we're required to, but based on the forum posts that seems to be the norm, especially since Dr. Lusth is rewarding 5 partial credit points if we can print the adjacency matrix.

However, the book says to use an adjacency list in its algorithm (on page 634 in line 8 it has G.Adj[u]. Page 590 says that this is the book's standard pseudocode for an adjacency list, not an adjacency matrix.) Moreover, I've read elsewhere (on the interwebs) that an adjacency list is the way to go for Prim's algorithm with a binomial (or Fibonacci) heap.

Am I missing something? Is there a reason we're using a matrix instead of a list? If I'm not mistaken, finding the lowest weighted edge from a vertex in an adjacency list takes much less time than finding it in an adjacency matrix, where you have you search through every index, one for each vertex, to find it, which is terribly slow in a sparsely populated matrix.

Although the adjacency matrix takes up unneeded space when the graph is sparse, Dr. Lusth is letting us use it. It provides constant look up times for finding the weight on an edge incident with 2 vertices. Also a 2D array has less complicated management than adj. lists.

The matrix can be used like an adjacency list for the algorithm by hard coding one index and having a variable iterate on the other index. I believe this takes the close to same amount of time (as long as the graph is not to sparse) if you use a adj. list or adj. matrix.

Got it. Just don't make a test case with an index of 1,000,000 like I did. You'll likely freeze your computer.