1. On an overnight camping trip in Sunnydale National Park, you are woken from a restless sleep by a scream. As you crawl out of your tent to investigate, a terrified park ranger runs out of the woods, covered in blood and clutching a crumpled piece of paper to his chest. As he reaches your tent, he gasps, "Get out... while... you...", thrusts the paper into your hands, and falls to the ground. Checking his pulse, you discover that the ranger is stone dead.

   You look down at the paper and recognize a map of the park, drawn as an undirected graph, where vertices represent landmarks in the park, and edges represent trails between those landmarks. (Trails start and end at landmarks and do not cross.) You recognize one of the vertices as your current location; several vertices on the boundary of the map are labeled EXIT.

   On closer examination, you notice that someone (perhaps the poor dead park ranger) has written a real number between 0 and 1 next to each vertex and each edge. A scrawled note on the back of the map indicates that a number next to an edge is the probability of encountering a vampire along the corresponding trail, and a number next to a vertex is the probability of encountering a vampire at the corresponding landmark. (Vampires can't stand each other's company, so you'll never see more than one vampire on the same trail or at the same landmark.) The note warns you that stepping off the marked trails will result in a slow and painful death.

   You glance down at the corpse at your feet. Yes, his death certainly looked painful. Wait, was that a twitch? Are his teeth getting longer? After driving a tent stake through the undead ranger's heart, you wisely decide to leave the park immediately.

   Describe and analyze an efficient algorithm to find a path from your current location to an arbitrary EXIT node, such that the total *expected number* of vampires encountered along the path is as small as possible. *Be sure to account for* ***both*** *the vertex probabilities* ***and*** *the edge probabilities!*

2. In this problem we will discover how you, too, can be employed by Wall Street and cause a major economic collapse! The *arbitrage* business is a money-making scheme that takes advantage of differences in currency exchange. In particular, suppose that 1 US dollar buys 120 Japanese yen; 1 yen buys 0.01 euros; and 1 euro buys 1.2 US dollars. Then, a trader starting with $1 can convert his money from dollars to yen, then from yen to euros, and finally from euros back to dollars, ending with $1.44! The cycle of currencies $\$ \to ¥ \to € \to \$$ is called an ***arbitrage cycle***. Of course, finding and exploiting arbitrage cycles before the prices are corrected requires extremely fast algorithms.

   Suppose n different currencies are traded in your currency market. You are given the matrix $R[1..n, 1..n]$ of exchange rates between every pair of currencies; for each i and j, one unit of currency i can be traded for $R[i, j]$ units of currency j. (Do *not* assume that $R[i, j] \cdot R[j, i] = 1$.)

   (a) Describe an algorithm that returns an array $V[1..n]$, where $V[i]$ is the maximum amount of currency i that you can obtain by trading, starting with one unit of currency 1, assuming there are no arbitrage cycles.

   (b) Describe an algorithm to determine whether the given matrix of currency exchange rates creates an arbitrage cycle.

   (c) Modify your algorithm from part (b) to actually return an arbitrage cycle, if it exists.

3. Let $G = (V, E)$ be a directed graph with weighted edges; edge weights could be positive, negative, or zero. In this problem, you will develop an algorithm to compute shortest paths between *every* pair of vertices. The output from this algorithm is a two-dimensional array $dist[1..V, 1..V]$, where $dist[i, j]$ is the length of the shortest path from vertex i to vertex j.

   (a) How could we delete some node v from this graph, without changing the shortest-path distance between any other pair of nodes? Describe an algorithm that constructs a directed graph $G' = (V', E')$ with weighted edges, where $V' = V \setminus \{v\}$, and the shortest-path distance between any two nodes in G' is equal to the shortest-path distance between the same two nodes in G. For full credit, your algorithm should run in $O(V^2)$ time.

   (b) Now suppose we have already computed all shortest-path distances in G'. Describe an algorithm to compute the shortest-path distances from v to every other node, and from every other node to v, in the original graph G. For full credit, your algorithm should run in $O(V^2)$ time.

   (c) Combine parts (a) and (b) into an algorithm that finds the shortest paths between *every* pair of vertices in the graph. For full credit, your algorithm should run in $O(V^3)$ time.

   The lecture notes (along with most algorithms textbooks and Wikipedia) describe a dynamic programming algorithm due to Floyd and Warshall that computes all shortest paths in $O(V^3)$ time. This is *not* that algorithm.