1. Suppose that you have just finished computing the array $dist[1..|V|, 1..|V|]$ of shortest-path distances between ***all*** pairs of vertices in an edge-weighted directed graph $G = (V, E)$. Unfortunately, you discover that you incorrectly entered the weight of a single edge $u \to v$, so all that precious CPU time was wasted. Or was it? Maybe your distances are correct after all!

   In each of the following problems, let $w(u \to v)$ denote the weight that you used in your distance computation, and let $w'(u \to v)$ denote the correct weight of $u \to v$. You may assume there are no negative weight cycles in G with either weight $w(u \to v)$ or $w'(u \to v)$.

   (a) Suppose $w(u \to v) > w'(u \to v)$; that is, the weight you used for $u \to v$ was *larger* than its true weight. Describe an algorithm that repairs the distance array in $O(n^2)$ ***time*** under this assumption.

   Solution: Let $dist'[1..n, 1..n]$ denote the *corrected* distance array we are trying to compute. Shortest paths contain each vertex at most once, and in particular a shortest path to u does not contain $u \to v$ and a shortest path *from* v does not contain $u \to v$ so $dist'[x, u] = dist[x, u]$ and $dist'[v, u] = dist[v, y]$.

   As stated in the hint, the shortest path from any vertex x to any vertex y either contains the edge $u \to v$ or it doesn't. We have $dist'[x, y] = dist[x, y]$ if the shortest path from x to y *doesn't* contain $u \to v$. Combined with the observation in the previous paragraph, we see

   $$dist'[x, y] = \min \left\{ \begin{array}{c} dist[x, y], \\ dist[x, u] + w'(u \to v) + dist[v, y] \end{array} \right\}.$$

   We can iterate over all pairs over vertices x and y applying the above formula in $O(|V|^2)$ ***time***. ■

   (b) Maybe even that was too much work. Describe an algorithm that determines whether your original distance array is actually correct in $O(1)$ ***time***, again assuming that $w(u \to v) > w'(u \to v)$.

   Solution: Let $dist'[1..n, 1..n]$ denote the *corrected* distance array we are trying to compute. If a shortest path from u to v under the fixed weights must use edge $u \to v$, then $dist'[u, v] = w'(u \to v) < dist[u, v]$. On the other hand, if any shortest path from u to v under the fixed weights *does not* use $u \to v$, then *any* path between any pair of vertices x and y using $u \to v$ can be changed to avoid $u \to v$ it without making the path longer, implying the original distance array already has the correct distances. In particular, $w'(u \to v) \geqslant dist[u, v]$.

   We check of $w'(u \to v) \geqslant dist[u, v]$. If so, the original distance array is correct. Otherwise, it is incorrect. ***The algorithm runs in $O(1)$ time.*** ■

(c) **To think about later:** Describe an algorithm that determines in $O(mn)$ ***time*** whether your distance array is actually correct, even if $w(u \to v) < w'(u \to v)$.

Solution: We claim the original distance array *dist* is incorrect if and only if there exists a vertex x such that $dist[x, v]$ is incorrect. The if direction is immediate. For the only if direction, suppose there exists a pair of vertices x and y such that $dist[x, y]$ is incorrect. Then, every shortest path from x to y under the weights w contains edge $u \to v$ implying the distance from x to v increases when reweighting $u \to v$. Let $dist'[1..n, 1..n]$ denote the *corrected* distance array we are trying to compute. We first compute all entries $dist'[x, v]$ in $O(|V||E|)$ time using Bellman-Ford in the reverse graph of G from v. For all $x \in V$, we compare $dist[x, v]$ to $dist'[x, v]$ and report distances are correct if every pair of values are equal. ***The algorithm takes $O(|V||E|)$ time.*** ■

(d) **To think about later:** Argue that when $w(u \to v) < w'(u \to v)$, repairing the distance array *requires* recomputing shortest paths from scratch, at least in the worst case.

Solution: Let $G' = (V', E')$ be a directed graph arbitrary positive edge weights $w : E' \to \mathbb{R}^+$.

Now, suppose $G = (V, E)$ is formed by adding two new vertices u and v to G' along with

- an edge $x \to u$ from each $x \in V'$ of weight $w(u \to s) = 0$,
- an edge $u \to v$ of weight $w(u \to v) = 0$, and
- an edge $v \to y$ to each $y \in V'$ of weight $w(v \to y) = 0$.

We have $dist[x, y] = 0$ for every pair of vertices $x, y \in V$, and no shortest path between a pair of vertices in G' uses an edge of G'.

Let M be any value larger than the largest vertex-to-vertex distance in G'. If we change the weight of $u \to v$ to $w'(u \to v) = M$, then every shortest path between a pair of vertices in G' will use only edges in G'; however, the previous paragraph implies we have no information about those paths, and we must compute them from scratch. ■

2. Suppose n different currencies are traded in your currency market. You are given a two-dimensional array $R[1..n, 1..n]$ containing 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 $MaxCurrency[1..n]$ where $MaxCurrency[1..n]$ 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.

Solution (graph reduction): We create a new directed graph $G = (V, E)$ with edge weights $w : E \to \mathbb{R}$ defined as follows:

- $V \coloneqq \{1, \ldots, n\}$.
- $E \coloneqq V \times V$.
- $w(i \to j) \coloneqq -\log_2 R[i, j]$.

Let $dist(i)$ denote the distance from vertex 1 to vertex i in G. We have $MaxCurrency[i] = 2^{-dist[i]}$.

Some weights in G may be negative, but there is no negative weight cycle, because the total weight of all cycles must be non-negative to avoid an arbitrage cycle. To complete the algorithm we call Bellman-Ford from vertex 1 and apply the equation above.

The algorithm runs in $O(|E||V|) = O(n^3)$ time. ■

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

Solution (graph reduction): We create the *same* directed graph $G = (V, E)$ with edge weights $w : E \to \mathbb{R}$ defined as follows:

- $V \coloneqq \{1, \ldots, n\}$.
- $E \coloneqq V \times V$.
- $w(i \to j) \coloneqq -\log_2 R[i, j]$.

There is an arbitrage cycle if and only if there is a negative weight cycle in G. To complete the algorithm we call the dynamic programming formulation of Bellman-Ford from any vertex s. Let $dist[1..n, 0..n]$ be the memoization array indexed by vertex and number of edges in a walk. We report there is an arbitrage cycle if and only if $dist[t, n] < dist[t, n-1]$ for some vertex t. Otherwise, we report there is no arbitrage cycle.

The algorithm runs in $O(|E||V|) = O(n^3)$ time. ■

(c) **To think about later:** Modify your algorithm from part (b) to actually return an arbitrage cycle if such a cycle exists.

Solution: We run the algorithm from the previous solution but do the following at the end instead of simply *reporting* that an arbitrage cycle exists. Fix any vertex t such that $dist[t,n] < dist[t,n-1]$. Edge $u \to v$ is the last one on the shortest walk from s to v with at most ℓ edges if and only if $dist[u,\ell-1] + w(u \to v) = dist[v,\ell]$, and there must be at least one such edge entering v. We trace an arbitrary walk $\langle t, w, \ldots \rangle$ along such edges using ℓ from each of $\langle n, n-1, \ldots, 0 \rangle$ marking each vertex as we go. A simple path has at most $n-1$ edges, so we must encounter an already marked vertex. Let v be the first already marked vertex encountered.

The subwalk from the first encounter of v to the second must have negative weight; otherwise, there would have been a walk from s to t of weight $dist[t,n]$ with strictly fewer than n edges, contradicting $dist[t,n] < dist[t,n-1]$. That negative weight subwalk is (the reverse of) our arbitrage cycle.

The algorithm runs in $O(|E||V|) = O(n^3)$ time. ■