ASSIGNMENT 6

DUE: Wednesday, March 16, 11:59 PM. DO NOT COPY. ACKNOWLEDGE YOUR SOURCES.
Please read http://www.student.cs.uwaterloo.ca/~cs341 for general instructions and policies.

Exercises. The following exercises are for practice only. You may ask about them in office hours. Do not hand them in.

1. Find a concrete example to show Dijkstra’s algorithm does not work when the graph has negative weights.

Problems.

1. [5 points] Wile E. Coyote (Coyote) is, as always, chasing the Road Runner (Runner). But for this question there are no roads. Both Runner and Coyote can only travel between \( n \) vertices of a network by running through one of \( m \) special one-way pipes labelled 1, 2, \ldots, \( m \). For each pipe we know the starting and ending vertex.

   The pipes are special because they affect the Coyote’s and Runner’s size. Each pipe has two different values \( C_i, R_i \in \mathbb{R}_{>0} \), which are the multipliers of the Coyote’s and Runner’s heights respectively, as they run through the pipe. For example, if the Runner and Coyote currently both have height 1, and both enter a pipe \( k \) with \( R_k = 0.6 \) and \( C_k = 1.2 \), they will emerge with heights 0.6 and 1.2, respectively: the Coyote is now twice as big as the Runner.

   The Runner starts running at a predetermined vertex \( s \). The Coyote starts at \( s \) an instant later, and will chase the Runner at the same speed as the Runner. You can essentially treat the Coyote and Runner as a single object, whose trajectory is entirely dictated by the Runner. At the start both the Runner and Coyote have size 1.

   Given the network (with no duplicate edges or self-loops) and the starting vertex \( s \), show how to determine in \( O((n + m)^{10}) \) time or faster if it is possible for the Runner to lead the Coyote through a sequence of pipes (possibly repeating) such that the Runner gets infinitely larger than the Coyote.

   Note: You may assume all arithmetic with real numbers is exact and that operations on real numbers can be done in constant time.

   Note: Relevant YouTube video: https://www.youtube.com/watch?v=KJiW7EF5aVk

2. You are given a weighted, directed graph \( G = (V, E) \) together with an integer \( h \geq 1 \). Your goal is to compute the minimum weight walks consisting of exactly \( h \) edges. That is, for all \( n^2 \) pairs of vertices \( u \) and \( v \), find the minimum weight of a walk consisting of \( h \) edges that starts at vertex \( u \) and ends at vertex \( v \) (or report \( \infty \) if no such walk exists between \( u \) and \( v \)).

   Note: As usual, \( V = \{1, 2, \ldots, n\} \), \( |E| = m \), and if \( (u, v) \in E \) then \( w(u, v) \) is the weight. You can also assume that there are no duplicate edges or self-loops.
(a) [10 points] Provide an $O(n^3h)$ time dynamic programming algorithm to compute the minimum weighted walk consisting of exactly $h$ edges between all pairs of vertices $i$ and $j$.

*Note:* Your solution must consist of the four parts mentioned in Assignment 4 regarding dynamic programming problems.

(b) [5 points] Let $A_{h_1}[u,v]$ be the $n \times n$ table containing the minimum weight of a walk from vertex $u$ to $v$ using exactly $h_1 \geq 1$ edges, $1 \leq u, v \leq n$. Note that $A_{h_1}[u,v] = \infty$ if no walk with $h_1$ edges from vertex $u$ to $v$ exists. Similarly, let $A_{h_2}[u,v]$ be the $n \times n$ table containing the minimum weight of a walk using exactly $h_2 \geq 1$ edges. Show how using just these two tables, and using $O(n^3)$ time, you can compute $A_{h_1+h_2}[u,v]$, the $n \times n$ table containing the minimum weight of a walk from vertex $u$ and $v$ that consists of exactly $h_1 + h_2$ edges.

*Note:* Your solution to this part question will be useful for Programming Assignment 2.

**Challenge Questions.** This is for fun and enrichment only. Do not hand it in.

1. There is a variant of Dijkstra called SPFA, which is instead of a priority queue, the improved distances are pushed into a queue. Show that on sparse graphs with $O(n)$ edges and weights in the range $[1, n]$, this algorithm can take up to $\Theta(n^2)$ time.