17 (100 PTS.) Avoiding negativity.

Let G be a directed graph with n vertices and m edges, with weights $w(\cdot)$ on the edges (weights on edges [here] can be any real number, including negative numbers). You are also given a start vertex s. Describe[1] an algorithm that outputs all the vertices x in G, such that *all* the walks from s to x in G do *not* contain a negative cycle.

We first use Kosaraju's algorithm to find all SCC's contained within the graph, as well as the graph's sources and sinks. Call this new graph of strongly connected components G'.
Runtime: O(m+n)

Find the SCC that contains the starting vertex s, where s ∈ V. That will be the new starting vertex s', where s' ∈ G'.

Create a new graph G'' = { V'', E'' }, where V'' and E'' are initially empty. Find all vertices reachable from s', and add them and their corresponding edges to V'' and E''.

Within each SCC, we will run the Bellman-Ford algorithm to find the shortest path from one vertex to the rest. If the Bellman-Ford algorithm detects a negative cycle, we will run this algorithm on the SCC:

> First, remove this SCC from V'', and remove the edges that point to the SCC. Next, for each edge that originates from the SCC, remove the edge and recursively call this algorithm on the nodes that the edge points to.

This is because all child nodes of the SCC have a path from the negative cycle, so they must be removed.

The SCC's that remain in G'' must be reachable from s, and they will contain all vertices x ∈ G, where all walks from s to x will not contain a negative cycle.

18 (100 PTS.) Controlled negativity.

Let G be a directed graph with n vertices and m edges, with weights $w(\cdot)$ on the edges.

18.A. (50 PTS.) Assume that for every vertex $v \in V(G)$ has a start price $\alpha(v)$ (not necessarily positive), and the weights on the edges are all positive. Describe[2] an algorithm that computes the length of the shortest path that ends at x (for all $x \in V(G)$). Here, a path π that starts at a vertex v and ends at x has ***length***

$$L(v, \pi) = \alpha(v) + w(\pi),$$

where $w(\pi)$ is the total weight of the edges of π. (Hint: First consider the case that $\alpha(\cdot)$ is strictly positive for all the vertices.) Prove that your algorithm is correct.

Let (u, v) mean a directed edge from vertex u to vertex v.

We know that every vertex v ∈ V(G) has a starting price a(v). Find a vertex vs ∈ V(G), where a(vs) ≤ a(vx), for all vx ∈ V(G). We will subtract the value of (a(vs) - 1) from the price of all v ∈ V(G). The subtraction of the minimum vertice value will make the redefined start price strictly positive.
Then we create a new starting vertex s, where a(s) = 0 and s has edges (s, v), where v ∈ V and the edge has weight w(v) = a(v). Call this new graph G'.

This allows us to find a path from s to x with the minimum length, where all edges have a positive weight.

This transformation has a runtime of O(n).

We can now apply Dijkstra's algorithm to G, with s as the starting node. This will give us the length of the shortest path from s to x.

Notice that the minimum length of all paths from s to v ∈ V, and then the minimum length from v to x in G' forms a bijection of the minimum length of the paths from all v ∈ V to x in G. Thus, we can find the correct minimum length L(v, π) = a(v) + w(π) with the modified formula L(s, π) = w(π) + (a(v_s) - 1)

Thus, given the length of the shortest path from s to x, we find the correct answer by adding (a(v_s) - 1) to the output of Dijkstra's algorithm for the given node x.

18 (100 PTS.) Controlled negativity.

Let G be a directed graph with n vertices and m edges, with weights $w(\cdot)$ on the edges.

18.B. (50 PTS.) Now, consider the variant where the weights on the edges can be negative or positive (but there is no start prices on the vertices). You are given in addition to G, two vertices $s, t \in V(G)$, and a parameter k. Describe[3] an algorithm that computes the shortest walk in G between s and t, where your walk is allowed to travel on at most k negative edges. What is the running time of your algorithm?

For credit, your solution needs to be polynomial in all parameters (n, m and k). Partial credit would be given for an efficient but suboptimal algorithm. The "fastest" algorithm seems to follow from using part (A).

Assumption: Since "the weights on the edges can be negative or positive", we will assume that it's impossible for an edge to have a weight of 0.

Define (u, v) to be a directed edge from vertex u to vertex v.
Define w(u, v) to be a function that returns the weight of (u, v).

We first construct a configuration graph G', that will be initialized as an empty graph. For each vertex v ∈ V(G), and each integer i, where 0 ≤ i ≤ k, add the vertex (v, i) to G'.

- v represents a vertex in V(G)
- i represents the number of negative edges that have already been traversed.

This means G' will have (k + 1) * n total vertices.

For each edge e ∈ E(G):
If w(u, v) < 0, then for all integers i, where 0 ≤ i ≤ (k - 1), add the directed edge ((u, i), (v, i+1)) to G'.
If w(u, v) > 0, then for all integers i, where 0 ≤ i ≤ k, add the directed edge ((u, i), (v, i)) to G'.
This means G' have approximately k * m total edges.

Notice that this configuration prevents the occurrence of a negative cycle, because all negative edges originating from (u, i) must point to some (v, i+1), and there does not exist a directed edge from (v, i+1) to (u, i).

We can now apply the Bellman-Ford algorithm, running it with the starting vertex (s, 0). Once the algorithm has finished, we return the minimum distance required to reach any vertex (t, i), where $0 \leq i \leq k$.

Runtime:
We create approximately k * n total vertices and k * m total edges. Thus, the runtime required for creating G' is O(kn + km)

The Bellman-Ford algorithm takes O(E + V) time. Thus, we get:
$O(E' * V') = O((kn) * (km)) = O(nmk^2)$

This means the total runtime of our algorithm is:
$O(kn + km) + O(nmk^2) = O(nmk^2)$