Let $G = (V, E, w)$ be a directed and weighted graph with edge weights $w(e)$, which can be negative or positive or zero, for each edge $e \in E$. You may assume that G has no *negatively weighted* cycles. Additionally, each vertex is coloured either *red* or *blue*. Recall that a *walk* is a sequence of edges joining a sequence of vertices; there are no restrictions on the edges or vertices (i.e. they may repeat or not repeat).

Given an integer $k \geq 2$, describe an $O(kn(km + n))$ algorithm that returns a walk with the smallest weight, satisfying the following conditions:

- The walk starts and ends on a vertex coloured red, and
- The number of vertices coloured blue in the walk is divisible by k.

Note. *For a directed and weighted graph $G = (V, E, w)$ where the edge weights can be negative, Bellman-Ford will compute the shortest distance from a vertex u to every vertex in time $O(|V| \cdot |E|)$. We will see how it works later but for now, you may use it as a black box.*

Hint. *Try constructing your edges in layers modulo k. You should have $km + 2n$ edges and $kn + 2$ vertices in your new graph.*

Solution. We can remodel the question with an auxilliary graph A, which consists of k layers (zero-indexed), each containing almost identical copies of G, except for the fact that all out-edges from blue vertices, instead of pointing to their neighbour in the current layer, they point to the equivalent of their neighbour one layer above, with blue vertices in the $(k-1)$th layer having out-edges pointing towards vertices in the 0th layer. This index of the layer you are currently in represents the number of blue vertices you have crossed so far modulo k.

We add a special End vertex, with 0-weighted inedges from all red vertices in layer 0. We should also have a special Begin vertex, which has 0-weighted out-edges pointing to every red vertex in layer 0. Applying Bellman-Ford's algorithm on A from the source vertex Begin, and finding the shortest path ending at vertex End, will give us our desired path.

The algorithm is correct because the layer you are currently in represents the number of blue vertices you have crossed so far modulo k, and so you can only reach the End vertex if your current path has a number of blue vertices divisible by k.

Without considering the extra nodes and vertices added due to the Begin and End vertices, there are km edges and kn vertices, as there are essentially k copies of G except that out-edges from blue vertices have been redirected. The addition the Begin and End introduces at most n new edges and exactly 1 new vertex each. Therefore the auxilliary graph A has at most $km + 2n$ edges and exactly $kn + 2$ vertices. Therefore the time complexity of this algorithm is

$$O(\textsc{numVertices}(A) \cdot \textsc{numEdges}(A)) = O((kn+2)(km+2n)) = O(kn(km+n)).$$