1. In class, we have seen Klein’s cycle cancelling algorithm for the Min Cost Circulation Problem (MCCP). This algorithm requires $O(m\log U)$ iterations in the worst case, i.e., its running time is not polynomial in $m$, log $C$ and log $U$. In this problem, we will see how to apply the idea of cost scaling on this algorithm to obtain an algorithm whose running time is polynomial in $m$, log $C$, and $U$.

(In fact, it is possible to apply the same idea on both costs and capacities to obtain an algorithm whose running time is polynomial in $m$, log $U$ and log $C$, but this is not required in this problem.)

Recall that in MCCP, a bidirected graph $G = (V, E)$, an anti-symmetric cost function $c : E \rightarrow \mathbb{Z}$, and a capacity function $u : E \rightarrow \mathbb{Z}$ are given. Let $n$ and $m$ denote the number of vertices and edges in $G$, $U = \max_{(v, w) \in E} |u(v, w)|$, and $C = \max_{(v, w) \in E} |c(v, w)|$.

(a) For every integer $i$, define the cost function $c^{(i)} : E \rightarrow \mathbb{Z}$ as follows:

$$c^{(i)}(v, w) := \text{sgn}(c(v, w)) \left\lfloor \frac{|c(v, w)|}{2^i} \right\rfloor,$$

where $\text{sgn}(x)$ is the sign of $x$. Notice that by the above definition, $c^{(0)}(v, w) = c(v, w)$ and $c^{\left\lceil \log(C+1) \right\rceil}(v, w) = 0$. Our objective is to find a way to solve MCCP for the cost function $c^{(i)}$, given its solution for $c^{(i+1)}$.

Let $f$ be an optimum circulation for $G$ with the cost function $c^{(i+1)}$ and the capacity function $u$. Prove that if we apply Klein’s cycle cancelling algorithm on $G$ with the cost function $c^{(i)}$ and capacity function $u$, starting from the circulation $f$, then the number of iterations of this algorithm is $O(mU)$.

(b) Use part (a) to obtain an algorithm for MCCP that requires $O(mU \log C)$ iterations.

2. Consider a directed graph $G = (V, E)$ with a length function $l : E \rightarrow \mathbb{Z}$ and a specified source vertex $s \in V$. The Bellman-Ford shortest path algorithm computes the shortest path lengths $d(v)$ between $s$ and every vertex $v \in V$, assuming that $G$ has no directed cycle of negative length (otherwise the problem is NP-hard). Here is a description of this algorithm:

The Bellman-Ford algorithm computes $d(v)$ by computing $d_k(v)$ = the shortest walk\(^1\) between $s$ and $v$ using exactly $k$ edges. $d_k(v)$ can be computed by the recurrence

$$d_k(v) = \min_{(w, v) \in E} \left[d_{k-1}(w) + l(w, v)\right].$$

\(^1\)A walk is like a path except that vertices might be repeated.
Let $h_1(v) = \min_{k=1,\ldots,d} d_k(v)$. It can be shown that if the graph has no negative cycle then $h_{n-1}(v) = d(v)$ for all $v \in V$. Moreover, the graph has no negative cycle iff, for all $v$, $d_n(v) \geq h_{n-1}(v)$.

(You are not required to prove any of the above facts.)

(a) Let $\mu^*$ be the minimum average length of a directed cycle $C$ of $G$, i.e.,

$$\mu^*(G) = \min_C \mu(C) = \min_C \frac{\sum_{(u,v) \in C} I(u,v)}{|C|}.$$ 

Using the Bellman-Ford algorithm, show how to compute $\mu^*$ in $O(nm)$ time.

(Hint: Use the fact that if we decrease the length of each edge by $\mu$ the average length of any cycle decreases by $\mu$.)

(b) Can you find the cycle $C$ with $\mu(C) = \mu^*$ using only $O(n^2)$ additional time? (In other words, suppose you are given all the values that the Bellman-Ford algorithm computes. Can you find a minimum mean cost cycle using this information in $O(n^2)$?)

3. Suppose we have $n$ objects that we want to store in a data structure. After storing these objects in the data structure, we would like to perform $m$ find operations on the data structure. Assume that the $i$th object is accessed $k_i \geq 1$ times. Therefore, $\sum_{i=1}^n k_i = m$. We want to evaluate the performance of the data structure by computing the total running time of these $m$ find queries (no other operation, such as delete or insert, is performed on the data structure).

(a) Show that if we store the objects in a splay tree, then no matter what the initial configuration of the splay tree is, and no matter in which order we access the objects, the total running time of the $m$ access operations is at most

$$O\left( m + \sum_{i=1}^n k_i \log \left( \frac{m}{k_i} \right) \right).$$

(b) Show that if we store the objects in a static binary search tree (i.e., a binary search tree that does not change by a find operation), then no matter in which order the objects are stored in the BST, and no matter in which order they are accessed, the total running time of the $m$ access operations is at least

$$\Omega\left( m + \sum_{i=1}^n k_i \log \left( \frac{m}{k_i} \right) \right).$$

This shows that the splay trees are within a constant factor as optimal as any static binary search tree.