COL351 Holi2023: Tutorial Problem Set 4

1. Let $G = (V, E)$ be a directed graph and let $w_e \geq 0$ be the weight of each edge $e \in E$. An embedding of G is a function $f : V \longrightarrow \mathbb{R}$ such that for each edge $e = (u, v)$, we have $f(v) - f(u) \leq w_e$. For vertices $s, t \in V$, the $s-t$-stretch of such an embedding is defined to be $f(t) - f(s)$. Design a polynomial time algorithm which, given a graph G, weights w_e for each edge of G, and vertices s, t of G, computes an embedding of G with maximum stretch.

2. Recall the Bellman-Ford algorithm discussed in class. Given an edge weighted-graph G and a vertex s of G, the algorithm either reports that the graph has a negative weight reachable from s, or finds the weight of a minimum-weight walk from s to every vertex v of G.

   We are more ambitious now. Given such a G and s, we would like to output an array W indexed by vertices of G such that for each vertex v, $W[v]$ is ∞ if v is not reachable from s, $-\infty$ if walks of arbitrarily small weight exist from s to v, and otherwise, $W[v]$ is the weight of a minimum weight s to v walk. Modify the Bellman-Ford to achieve this.

3. Recall that while discussing the Floyd-Warshall algorithm in class, we assumed that the given graph does not have a negative weight cycle. Modify the algorithm so that it works even in the presence of negative weight cycles. Specifically for every pair u, v of vertices, your algorithm should report ∞ if v is not reachable from u, $-\infty$ if walks of arbitrarily small weight exist from u to v, and the length of a minimum weight u to v walk otherwise.

4. Let G be an edge-weighted directed graph without non-positive weight cycles, and s be a vertex of S. Design a polynomial-time algorithm that, given G, the weights of its edges, and s, outputs the number of minimum-weight paths from s to each vertex v of G.

5. You have a huge amount of money in some currency (say USD) which you want to convert into another (say INR). While you could convert your USDs to INRs directly, it could potentially be more beneficial to convert your USDs to EUROs, then to AUDs, and then to INRs, considering the rates of conversion available. In this problem, you are asked to design an algorithm that finds the optimal sequence of conversions you should perform. Formally, you are given a set of n currencies – $C_1, \ldots, C_n$. You have 1 unit of money in currency C_1, and you wish to convert it to currency C_n. You are also given a set of available currency change rates $r(i, j) > 0$ for some m pairs (C_i, C_j) of currencies, which means you can potentially change x units of C_i to $x \cdot r(i, j)$ units of C_j. Design a strongly polynomial-time algorithm to compute the maximum possible amount that you can obtain in currency C_n by a sequence of currency changes starting from 1 unit of C_1.

6. You must have heard of the traveling salesperson problem (TSP), in which you are given an edge-weighted directed graph G, and you are required to compute (the weight of) a minimum-weight cycle that passes through all vertices of the graph. The obvious brute-force algorithm is to try all possible permutations of the n vertices, and this takes $2^{\Omega(n \log n)}$ time. Use your dynamic-programming skills to design a faster algorithm – one that runs in time $2^{O(n)}$. (The solution should be readily available on the internet and in textbooks. If you see it, you lose one opportunity to practice dynamic programming problems.)