HW 2

Due on **Wednesday, September 8, 2021 at 10am**

CS 473: Algorithms, Fall 2021 Version: **1.1**

Submission guidelines and policies as in homework 1.

4 (100 PTS.) Good path.

Consider a DAG G with n vertices and m edges. Each vertex v of G has an associated value $\alpha_v \geq 0$. (To solve this problem, you might want to revisit topological ordering, and how to compute it in linear time.)

4.A. (20 PTS.) The ***value*** of a path π in G is $\mathrm{val}(\pi) = \sum_{v \in V(\pi)} \alpha_v$. Show an algorithm that, in linear time, computes a path π in G of maximum value. (We remind the reader that linear time for a graph, means linear time in the number of edges and vertices in the graph.)

Show how to use this algorithm to compute the longest path (i.e., path with most edges) in G in linear time.

4.B. (20 PTS.) Assume that there are k paths (not necessarily disjoint) that cover all the vertices of G. Describe an algorithm that computes a path in G of value $\geq \mathrm{val}(G)/k = \sum_{v \in V(G)} \alpha_v / k$ in linear time, and prove that the returned path has the desired property.

4.C. (40 PTS.) Assume that there are k paths (not necessarily disjoint) that cover all the vertices of G. Describe an algorithm, as fast as possible, that computes $O(k \log n)$ paths that cover all the vertices of G. (Hint: Use previous part repeatedly, adapting the values of vertices that are covered by a just computed path after each iteration.) Prove the bound on the number of paths computed (hint: Argue that after computing $O(k)$ paths, at least half the vertices in the graph are covered).

4.D. (20 PTS.) You are given a positive integer k, and an oracle, such that given two vertices u, v of G as query, the oracle returns either (u, v) or (v, u), such that if you add the returned edge to G it remains a DAG. Furthermore, this oracle keep working in this fashion for any number of such edges added to the DAG. (Thus, if you call the oracle on all the pairs of vertices in G, you would get a DAG where all the pair of vertices are connected by an edge. Then, a path of length $n - 1$ exists.)

Let π be the longest path in G. Describe an algorithm, performing some oracle queries (the fewer the better), such that in the resulting DAG G' there is a path of length $|\pi| + k$, and your algorithm outputs this (longer) path. How many oracle queries does your algorithm performs? What is the running time of your algorithm?

5 (100 PTS.) Some NP-COMPLETEness, despite everything.

5.A. (50 PTS.) You are given a directed graph G with n vertices and m edges, and weights on the edges (the weights can be negative). Given two vertices s, t of G prove that deciding if there is a simple path (i.e., no vertex is repeated more than once) between s and t of price exactly zero is NP-COMPLETE.

(I.e., prove that the problem is in NP, and polynomially reduce one of the known NP-HARD/NP-COMPLETE problems to this problem.)

5.B. (50 PTS.) Show a polynomial time reduction from 3DM to Subset Sum (potentially via Vector Subset Sum). Prove that your reduction is correct.

See class notes for the definition of these problems:

`https://courses.engr.illinois.edu/cs473/fa2021/lec/notes/03_npc_III.pdf`.

6 (100 PTS.) Jump, jump, jump!

You are give an directed graph G with n vertices and m edges, with positive prices on the edges of G. The vertices have colors. You are allowed to jump for free from a vertex to any another vertex of the same color. Assume the colors used on the vertices are $[\![t]\!] = \{1, 2, \ldots, t\}$, with a color of a vertex $v \in \mathsf{V}(\mathsf{G})$ being $c(v) \in [\![t]\!]$.

You are given a parameter $k \leq n$, and two vertices s and t. The task at hand is to compute the shortest path in G between s and t when you are allowed to use at most k jumps. Describe a polynomial time reduction from this problem to Dijkstra – namely, your algorithm should construct a graph H such that one can solve the problem via single invocation of Dijkstra on the graph H. What is the running time of your algorithm as a function of n, m, k and t. The faster the better. How many edges and vertices the constructed graph has (the fewer the better, naturally).