HW 8

Due on **Wednesday, October 30, 2024 at 9am CST**

CS/ECE 374A: Intro. Algorithms & Models of Computation, Fall 2024 Version: **1.3**

Submission instructions as in previous homeworks.

Extra guidelines for no extra charge. As a reminder, describing an algorithm requires not only describing the algorithm itself, but you also have to explain how and why your algorithm works. You also need to provide an analysis of the running time of your algorithm. Finally, you also need to explain the correctness of your algorithm, and provide a proof if asked to provide one.
For information how the running time of your algorithm is graded, see the FAQ.

15 (100 PTS.) Traveling in tolls.

You are given a directed graph G with n vertices and $m \geq n$ edges. You are also given a source vertex s. Each edge $e \in \mathsf{E}(\mathsf{G})$ is labeled by an integer number $\ell(e) \in [\![k]\!] = \{1, \ldots, k\}$ (let ***i*-edge** denote an edge with label i). Before traveling from s, you need to pay a toll – there are $k \leq n$ different tolls you can buy, with the ith toll having price (say) i. If you bought the ith toll, then you can travel on all edges with their label being at most i (and you can not travel on any of the edges marked by $i+1, \ldots, k$).

A vertex is ***i*-reachable** if the cheapest toll you need to buy to reach it (from s) is i. (Assume for simplicity that all the vertices of G are reachable from s.) All the vertices that are i-reachable, are in the ith zone, denoted by $Z_i \subseteq \mathsf{V}(\mathsf{G})$.

15.A. (30 PTS.) Describe *shortly* an algorithm that computes the zones $Z_1, \ldots, Z_k$. For ***credit***, the overall running time of your algorithm should be $O((n+m)k)$. (You do not need to provide pseudo-code for this part.)

15.B. (70 PTS.) A more sophisticated algorithm works in stages – first it computes Z_1 by performing **BFS** using only 1-edges. In the second stage, it resumes the **BFS**, and computes all the vertices in Z_2 (importantly, the algorithm might encounter new 1-edges that should be explored during this stage). The algorithm continues in the same fashion – in the ith stage is computes Z_i (conceptually by "resuming" the **BFS** done in the $(i-1)$th stage). Describe in detail how to implement this variant of **BFS**. For ***credit***, your algorithm must run in $O(n+m)$ time overall, and you need to provide detailed pseudo-code (in addition to a detailed description in plain English how your algorithm works), and explain in detail why your algorithm is correct. Also, provided a detailed explanation for the running time of the algorithm.

16 (100 PTS.) Musical plugs: The robots uprising.

You are given a directed graph $\mathsf{G} = (\mathsf{V}, \mathsf{E})$ with n vertices and m edges. Each vertex v of G is labeled $\ell(v) \in \{C, D, E, F, G, A, B\}$, one of the seven notes in the C Major scale. Three robots are placed on the vertices. In each round, a robot plays the note specified by the label of its current location, and then moves to a neighboring vertex (it must move). Thus, the three robots play their respective notes, and move in sync. The can potentially play up to $7^3 = 343$ different combinations, but among them, seven combinations are ***chords***: $C_1 = (C, E, G)$, $C_2 = (D, F, A)$, $C_3 = (E, G, B)$, $C_4 = (F, A, C)$, $C_5 = (G, B, D)$, $C_6 = (A, C, E)$, $C_7 = (B, D, F)$. A ***configuration*** is a triple

$(v_1, v_2, v_3) \in \mathsf{V} \times \mathsf{V} \times \mathsf{V}$ designating the locations of the three robots. A configuration is ***chordal*** if $(\ell(v_1), \ell(v_2), \ell(v_3)) \in \{C_1, \ldots, C_7\}$.

So, you have to place three robots at three vertices of G in the beginning of the process. At each point in time they have to play a chord and move (if they get stuck, and can not play a chord, the game is over). A ***symphony*** is a sequence of such valid configurations (configurations might repeat). The quality of the symphony is the number of unique configurations in the symphony. Describe an algorithm, as fast as possible, that computes the maximum quality symphony – your algorithm should output the start configuration, and the quality of the symphony – it does not need to output the symphony itself.

(Hint: SCC.)