CS/ECE 374 A (Spring 2022)
Homework 11 (due April 28 Thursday at 10am)

Instructions: As in previous homeworks.

Problem 11.1: Consider the following problem COLORFUL-WALK (which is an extension of Problem 8.2(b) from HW8):

> *Input*: a directed graph $G = (V, E)$ with n vertices and m edges, a vertex $s \in V$, and a color $c(e) \in \{1, \ldots, k\}$ for each edge $e \in E$, and a number ℓ.
>
> *Output*: "yes" iff there exists a walk that starts at s and encounters at least ℓ distinct colors.

Consider the SET-COVER problem:

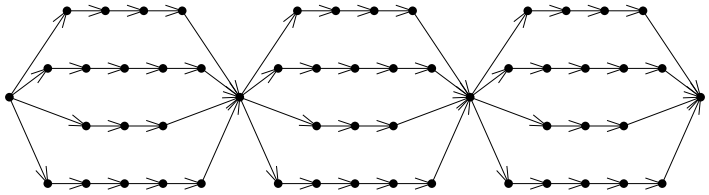
> *Input*: a set X of N elements, a collection of M subsets $\mathcal{S} = \{S_1, \ldots, S_M\}$ where each $S_i \subseteq X$, and a number K.
>
> *Output*: "yes" iff there exists a subcollection $\mathcal{T} \subseteq \mathcal{S}$ of K subsets, such that the union of the subsets in $\mathcal{T}$ includes all elements of X.

Describe a polynomial-time reduction from SET-COVER to COLORFUL-WALK. Follow these steps:

- Given an input to SET-COVER (i.e., X, $\mathcal{S}$, and K), describe a construction of an input to COLORFUL-WALK (i.e., G, s, $c(\cdot)$, and ℓ). Check that your construction takes polynomial time.
- Prove that the input to SET-COVER has a "yes" answer *if and only if* your input to COLORFUL-WALK has a "yes" answer.

Since SET-COVER is a well-known NP-hard problem, it would then follow that COLORFUL-WALK is NP-hard.

(Hint: for your construction of the directed graph G, try something like the picture below, with appropriate colors assigned to edges. Remember that in the actual description of your construction, you are given X, $\mathcal{S}$, and K; you are *not* given $\mathcal{T}$, which may or may not exist, since the goal is to determine whether the answer is "yes" or "no".)



Problem 11.2: By now, everyone has heard of Wordle, the online word-guessing game. Here, you will consider a tougher variant of the game, where the word length n is a variable, all strings of length n from a fixed alphabet are legal words (no dictionary needed!), and after each guess, you are only told whether there is a green (matching) position, but not the number of green (nor yellow) positions, nor where they are exactly. You will show that in this version of the game, just deciding whether there exists a solution after a series of guesses is already hard (so forget about the question of how to generate a good next guess!).

If you didn't follow the above paragraph, don't worry—here is the precise definition of our problem: For two strings y and z of length n, first define match(y, z) to be true if there exists a position k such that the k-th symbol of y is equal to the k-th symbol of z, and false otherwise. The statement of the Tough-Wordle problem is as follows:

> *Input*: a finite alphabet Σ, a list of m strings $y_1, \ldots, y_m \in \Sigma^n$, and m Boolean values $b_1, \ldots, b_m \in \{\text{true}, \text{false}\}$.
> *Output*: "yes" iff there exists a string $z \in \Sigma^n$ such that match$(y_i, z) = b_i$ for each $i = 1, \ldots, m$.

(Example: on the input with $\Sigma = \{A, B, C\}$, $y_1 = AAAA$, $b_1 = \text{true}$, $y_2 = BBCC$, $b_2 = \text{true}$, $y_3 = BCAB$, $b_3 = \text{false}$, the answer is "yes", e.g., by choosing $z = ABCC$.)

Describe a polynomial-time reduction from 3SAT to Tough-Wordle. Follow these steps:

- Given an input to 3SAT (i.e., a Boolean formula F in 3CNF form), describe a construction of an input to Tough-Wordle (i.e., an alphabet Σ, and a list of strings and Boolean values). Check that your construction takes polynomial time.
- Prove that F is satisfiable *if and only if* your input to Tough-Wordle has a "yes" answer.

Since 3SAT is NP-hard, it would then follow that Tough-Wordle is NP-hard.

(Hint: in your construction, an alphabet of size 3 (0, 1, and an extra symbol) would be sufficient. Suppose F has n variables and m clauses. Intuitively, the string z (if exists) should correspond to a satisfying assignment of F (if exists). For each clause C_i, construct a string y_i of length n (and a corresponding Boolean value b_i) to somehow make sure that z contains at least one true literal of C_i. Also, construct an extra string to somehow make sure that z uses only 0s and 1s and not the extra symbol... Remember that in the actual description of your construction, all you are given is F; you are *not* given a satisfying assignment, which may or may not exist, since the goal is to determine whether the answer is "yes" or "no".)