1. Suppose you are given a magic black box that somehow answers the following decision problem *in polynomial time*:

   - Input: A directed graph G and a positive integer L. (The edges of G are not weighted, and G is not necessarily a dag.)
   - Output: True if G contains a (simple) path of length L, and False otherwise.[1]

   (a) Using this black box as a subroutine, describe algorithms that solves the following optimization problem *in polynomial time*:

   - Input: A directed graph G.
   - Output: The length of the longest path in G.

   (b) Using this black box as a subroutine, describe algorithms that solves the following search problem *in polynomial time*:

   - Input: A directed graph G.
   - Output: The longest path in G

   [Hint: You can use the magic box more than once.]

2. An ***independent set*** in a graph G is a subset S of the vertices of G, such that no two vertices in S are connected by an edge in G. Suppose you are given a magic black box that somehow answers the following decision problem *in polynomial time*:

   - Input: An undirected graph G and an integer k.
   - Output: True if G has an independent set of size k, and False otherwise.[2]

   (a) Using this black box as a subroutine, describe algorithms that solves the following optimization problem *in polynomial time*:

   - Input: An undirected graph G.
   - Output: The size of the largest independent set in G.

   (b) Using this black box as a subroutine, describe algorithms that solves the following search problem *in polynomial time*:

   - Input: An undirected graph G.
   - Output: An independent set in G of maximum size.

   [Hint: You can use the magic box more than once.]

[1] You already know how to solve this problem in polynomial time *when the input graph G is a dag*, but this magic box works for *every* input graph.

[2] It is not hard to solve this problem in polynomial time via dynamic programming when the input graph G is a *tree*, but this magic box works for *every* input graph.

To think about later:

3. Formally, a ***proper coloring*** of a graph $G = (V, E)$ is a function $c \colon V \to \{1, 2, \ldots, k\}$, for some integer k, such that $c(u) \neq c(v)$ for all $uv \in E$. Less formally, a valid coloring assigns each vertex of G a color, such that every edge in G has endpoints with different colors. The ***chromatic number*** of a graph is the minimum number of colors in a proper coloring of G.

   Suppose you are given a magic black box that somehow answers the following decision problem *in polynomial time*:

   - INPUT: An undirected graph G and an integer k.
   - OUTPUT: TRUE if G has a proper coloring with k colors, and FALSE otherwise.[3]

   Using this black box as a subroutine, describe an algorithm that solves the following ***coloring problem*** *in polynomial time*:

   - INPUT: An undirected graph G.
   - OUTPUT: A valid coloring of G using the minimum possible number of colors.

   *[Hint: You can use the magic box more than once. The input to the magic box is a graph and **only** a graph, meaning **only** vertices and edges.]*

4. Suppose you are given a magic black box that somehow answers the following decision problem in *polynomial time*:

   - INPUT: A boolean circuit K with n inputs and one output.
   - OUTPUT: TRUE if there are input values $x_1, x_2, \ldots, x_n \in \{\text{TRUE}, \text{FALSE}\}$ that make K output TRUE, and FALSE otherwise.

   Using this black box as a subroutine, describe an algorithm that solves the following related search problem *in polynomial time*:

   - INPUT: A boolean circuit K with n inputs and one output.
   - OUTPUT: Input values $x_1, x_2, \ldots, x_n \in \{\text{TRUE}, \text{FALSE}\}$ that make K output TRUE, or NONE if there are no such inputs.

   [Hint: You can use the magic box more than once.]

[3]Again, it is not hard to solve this problem in polynomial time via dynamic programming when the input graph G is a *tree*, but this magic box works for *every* input graph.