1. 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 $x_1, x_2, \ldots, x_n \in \{\text{TRUE, FALSE}\}$ 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 $x_1, x_2, \ldots, x_n \in \{\text{TRUE, FALSE}\}$ that make $K$ output TRUE, or None if there are no such inputs.

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

**Solution:** For any boolean circuit $K$ with inputs $x_1, \ldots, x_n$, let $K \land x_i$ be the boolean circuit obtained from $K$ by adding a new AND gate, with one input connected to the output of $K$ and the other to the input $x_i$. Similarly, let $K \land \overline{x_i}$ be the boolean circuit obtained from $K$ by adding a NOR gate, with input connected to $x_i$, and an AND gate, with one input connected to the output of $K$ and the other to the NOR gate. For both of these circuits, the output of the new AND gate is the output of the circuit.

Suppose CircuitSat($K$) returns TRUE if $K$ is satisfiable and FALSE otherwise. Then the following algorithm constructs a satisfying input assignment for $K$ or correctly reports that no such assignment exists.

```python
SatAssignment(K):
    if CircuitSat(K) = False
        return None
    for i ← 1 to n
        if CircuitSat(K' ∧ x_i)
            K ← K ∧ x_i
            A[i] ← True
        else
            K ← K ∧ \overline{x_i}
            A[i] ← False
    return A[1..n]
```

The correctness of this algorithm follows by induction from the following observation:

**Claim 1.** The circuit $K \land x_i$ is satisfiable if and only if $K$ has a satisfying assignment where $x_i = \text{TRUE}$.

**Proof:** First, if $K \land x_i$ has a satisfying assignment, then that input assignment must satisfy $K$ and must have have $x_i = \text{TRUE}$, because otherwise the AND gate would output FALSE.
On the other hand, if \( K \) has a satisfying assignment where \( x_i = \text{TRUE} \), then that input assignment also satisfies \( K \land x_i \), because that’s how AND gates do. □

The algorithm runs in polynomial time. Specifically, suppose \( \text{CIRCUITSat}(K) \) runs in \( O(N^c) \) time, where \( N \) is the total number of vertices and edges in dag representing \( K \). (The vertices consist of the inputs, the internal gates, and the output; the edges are the wires between those points.) Then \( \text{SATAssignment}(K) \) runs in time

\[
O(N^c) + \sum_{i=1}^{n} O((N + 5i)^c) \leq (N + 1) \cdot O((6N)^c) = O(N^{c+1}),
\]

which is polynomial in \( N \). ■
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**: \( \text{TRUE} \) if \( G \) has an independent set of size \( k \), and \( \text{FALSE} \) otherwise.

(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 \( G \).

[Hint: You've seen this problem before.]

**Solution**: Suppose \( \text{IndSet}(V, E, k) \) returns \( \text{TRUE} \) if the graph \( (V, E) \) has an independent set of size \( k \), and \( \text{FALSE} \) otherwise. Then the following algorithm returns the size of the largest independent set in \( G \):

\[
\text{MaxIndSetSize}(V, E):
\]

\[
\text{for } k \leftarrow 1 \text{ to } V \\
\text{if } \text{IndSet}(V, E, k + 1) = \text{FALSE} \\
\text{return } k
\]

A graph with \( n \) vertices cannot have an independent set of size larger than \( n \), so this algorithm must return a value. If \( G \) has an independent set of size \( k \), then it also has an independent set of size \( k - 1 \), so the algorithm is correct.

The algorithm clearly runs in polynomial time. Specifically, if \( \text{IndSet}(V, E, k) \) runs in \( O((V + E)^c) \) time, then \( \text{MaxIndSetSize}(V, E) \) runs in \( O((V + E)^{c+1}) \) time.

Yes, we could have used binary search instead of linear search. Whatever. ■
(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.

**Solution (delete vertices):** I’ll use the algorithm $\text{MaxIndSetSize}(V,E)$ from part (a) as a black box instead. Let $G - v$ denote the graph obtained from $G$ by deleting vertex $v$, and let $G - N(v)$ denote the graph obtained from $G$ by deleting $v$ and all neighbors of $v$.

\[
\begin{align*}
\text{MaxIndSet}(G) & : \\
S & \leftarrow \emptyset \\
k & \leftarrow \text{MaxIndSetSize}(G) \\
\text{for all vertices } v \text{ of } G & : \\
\text{if } \text{MaxIndSetSize}(G - v) = k & : \\
G & \leftarrow G - v \\
\text{else} & : \\
G & \leftarrow G - N(v) \\
\text{add } v \text{ to } S
\end{align*}
\]

Return $S$

Correctness of this algorithm follows inductively from the following claims:

**Claim 2.** $\text{MaxIndSetSize}(G - v) = k$ if and only if $G$ has an independent set of size $k$ that excludes $v$.

**Proof:** Every independent set in $G - v$ is also an independent set in $G$; it follows that $\text{MaxIndSetSize}(G - v) \leq k$.

Suppose $G$ has an independent set $S$ of size $k$ that does excludes $v$. Then $S$ is also an independent set of size $k$ in $G - v$, so $\text{MaxIndSetSize}(G - v)$ is at least $k$, and therefore equal to $k$.

On the other hand, suppose $G - v$ has an independent set $S$ of size $k$. Then $S$ is also a maximum independent set of $G$ (because $|S| = k$) that excludes $v$. □

The algorithm clearly runs in polynomial time.
Solution (add edges): I’ll use the algorithm \textsc{MaxIndSetSize}(V, E) from part (a) as a black box instead. Let \( G + uv \) denote the graph obtained from \( G \) by adding edge \( uv \).

\begin{verbatim}
\textbf{MaxIndSet(G)}:
\begin{algorithmic}
  \State \( k \leftarrow \text{MaxIndSetSize}(G) \)
  \If {\( k = 1 \)}
    \State return any vertex
  \Else
    \For {all vertices \( u \)}
      \For {all vertices \( v \)}
        \If {\( u \neq v \) and \( uv \) is not an edge}
          \If {\( \text{MaxIndSetSize}(G + uv) = k \)}
            \State \( G \leftarrow G + uv \)
          \EndIf
        \EndIf
      \EndFor
    \EndFor
    \State \( S \leftarrow \emptyset \)
    \For {all vertices \( v \)}
      \If {\( \deg(v) < V - 1 \)}
        \State add \( v \) to \( S \)
      \EndIf
    \EndFor
  \EndIf
\EndAlgorithm
\end{verbatim}

The algorithm adds every edge it can without changing the maximum independent set size. Let \( G' \) denote the final graph. Any independent set in \( G' \) is also an independent set in the original input graph \( G \). Moreover, the largest independent set in \( G' \) is also a largest independent set in \( G \). Thus, to prove the algorithm correct, we need to prove the following claims about the final graph \( G' \):

\textbf{Claim 3.} The maximum independent set in \( G' \) is unique.

\textbf{Proof:} Suppose the final graph \( G' \) has more than two maximum independent sets \( A \) and \( B \). Pick any vertex \( u \in A \setminus B \) and any other vertex \( v \in A \). The set \( B \) is still an independent set in the graph \( G' + uv \). Thus, when the algorithm considered edge \( uv \), it would have added \( uv \) to the graph, contradicting the assumption that \( A \) is an independent set. \( \square \)

\textbf{Claim 4.} Suppose \( k > 1 \). The unique maximum independent set of \( G' \) contains vertex \( v \) if and only if \( \deg(v) < V - 1 \).

\textbf{Proof:} Let \( S \) be the unique maximum independent set of \( G' \), and let \( v \) be any vertex of \( G \). If \( v \in S \), then \( v \) has degree at most \( V - k < V - 1 \), because \( v \) is disconnected from every other vertex in \( S \).

On the other hand, suppose \( \deg(v) < V - 1 \) but \( v \notin S \). Then there must be at least vertex \( u \) such that \( uv \) is not an edge in \( G' \). Because \( v \notin S \), the set \( S \) is still an independent set in \( G' + uv \). Thus, when the algorithm considered edge \( uv \), it would have added \( uv \) to the graph, and we have a contradiction. \( \square \)

The algorithm clearly runs in polynomial time.

\textbf{□}
To think about later:

3. Formally, a **proper coloring** of a graph \( G = (V,E) \) is a function \( c : V \to \{1,2,\ldots,k\} \), 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**: \( \text{TRUE} \) if \( G \) has a proper coloring with \( k \) colors, and \( \text{FALSE} \) otherwise.

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.*

**Solution:** First we build an algorithm to compute the minimum number of colors in any valid coloring.

```plaintext
CHROMATICNUMBER(G):
for k ← V down to 1
  if COLORABLE(G, k − 1) = FALSE
    return k
```

Given a graph \( G = (V,E) \) with \( n \) vertices \( v_1, v_2, \ldots, v_n \), the following algorithm computes an array \( \text{color}[1..n] \) describing a valid coloring of \( G \) with the minimum number of colors.

[continued on next page]
In any $k$-coloring of $H$, the new vertices $z_1, \ldots, z_k$ must have $k$ distinct colors, because every pair of those vertices is connected. We assign $\text{color}[i] \leftarrow c$ to indicate that there is a $k$-coloring of $H$ in which $v_i$ has the same color as $z_c$. When the algorithm terminates, $\text{color}[1..n]$ describes a valid $k$-coloring of $G$.

To prove that the algorithm is correct, we must prove that for all $i$, when the $i$th iteration of the outer loop ends, $G$ has a valid $k$-coloring that is consistent with the partial coloring $\text{color}[1..i]$. Fix an integer $i$. The inductive hypothesis implies that when the $i$th iteration of the outer loop begins, $G$ has a $k$-coloring consistent with the first $i-1$ assigned colors. (The base case $i = 0$ is trivial.) If we connect $v_i$ to every new vertices except $z_c$, then $v_i$ must have the same color as $z_c$ in any valid $k$-coloring. Thus, the call to \text{Colorable} inside the inner loop returns True if and only if $H$ has a $k$-coloring in which $v_i$ has the same color as $z_c$ (and the previous $i-1$ vertices are also colored). So \text{Colorable} must return True during the second inner loop, which completes the inductive proof.

This algorithm makes $O(kn) = O(n^2)$ calls to \text{Colorable}, and therefore runs in polynomial time.

\[\text{Coloring}(G)\]  
\[k \leftarrow \text{ChromaticNumber}(G)\]  
\[H \leftarrow G\]  
\[\text{for } c \leftarrow 1 \text{ to } k\]  
\[\text{add vertex } z_c \text{ to } G\]  
\[\text{for } i \leftarrow 1 \text{ to } c-1\]  
\[\text{add edge } z_i z_c \text{ to } H\]  
\[\text{for } i \leftarrow 1 \text{ to } n\]  
\[\text{for } c \leftarrow 1 \text{ to } k\]  
\[\text{add edge } v_i z_c \text{ to } H\]  
\[\text{for } c \leftarrow 1 \text{ to } k\]  
\[\text{remove edge } v_i z_c \text{ from } H\]  
\[\text{if } \text{Colorable}(H, k) = \text{True}\]  
\[\text{color}[i] \leftarrow c\]  
\[\text{break inner loop}\]  
\[\text{add edge } v_i z_c \text{ from } H\]  
\[\text{return color}[1..n]\]