1. Let $G = (V, E)$ be a graph. A set of edges $M \subseteq E$ is said to be a matching if no two edges in $M$ intersect at a vertex. A matching $M$ is perfect if every vertex in $V$ is incident to some edge in $M$; alternatively $M$ is perfect if $|M| = |V|/2$ (which in particular implies $|V|$ is even).

The PERFECTMATCHING problem is the following: does the given graph $G$ have a perfect matching?

This can be solved in polynomial time which is a fundamental result in combinatorial optimization with many applications in theory and practice. It turns out that the PERFECTMATCHING problem is easier to solve in bipartite graphs. A graph $G = (V, E)$ is bipartite if its vertex set $V$ can be partitioned into two sets $L, R$ (left and right say) such that all edges are between $L$ and $R$ (in other words $L$ and $R$ are independent sets). Here is an attempted reduction from general graphs to bipartite graphs.

Given a graph $G = (V, E)$ create a bipartite graph $H = (V \times \{1, 2\}, E_H)$ as follows. Each vertex $u$ is made into two copies $(u, 1)$ and $(u, 2)$ with $V_1 = \{(u, 1) \mid u \in V\}$ as one side and $V_2 = \{(u, 2) \mid u \in V\}$ as the other side. Let $E_H = \{((u, 1), (v, 2)) \mid (u, v) \in E\}$. In other words we add an edge between $(u, 1)$ and $(v, 2)$ iff $(u, v)$ is an edge in $E$. Note that $\{(u, 1), (u, 2)\}$ is not an edge in $H$ for any $u \in V$ since there are no self-loops in $G$. Is the preceding reduction correct? To prove it is correct we need to check that $H$ has a perfect matching if and only if $G$ has one.

(a) Prove that if $G$ has perfect matching then $H$ has a perfect matching.

(b) Consider $G$ to be the complete graph on 3 vertices (a triangle). Show that $G$ has no perfect matching but $H$ has a perfect matching.

(c) Extend the previous example to obtain a graph $G$ with an even number of vertices such that $G$ has no perfect matching but $H$ has one.

Thus the reduction is incorrect although one of the directions is true.

Solution: (a) Suppose $M$ is a perfect matching in $G$. We construct a perfect matching $M’$ in $H$ as follows. For each edge $(u, v) \in M$ include the edges $((u, 1), (v, 2))$ and $((v, 1), (u, 2)) \in M’$. It remains to verify that $M’$ is a perfect matching. For each vertex $u \in V$ there is exactly one edge $(u, v) \in M$ incident to $u$ since $M$ is perfect. This means that in $H$ there is exactly one edge $((u, 1), (v, 2))$ incident to vertex $(u, 1)\in M’$ and also exactly one edge $((v, 1), (u, 2))$ incident to $(u, 2)$. Thus $M’$ is a perfect matching in $H$.

(b) Suppose the three vertices of $G$ are labeled $a, b, c$. The edges of $G$ are $(a, b), (b, c), (c, a)$. Then in $H$ one can easily verify that the set of edges $\{((a, 1), (b, 2)), ((b, 1), (c, 2)), ((c, 1), (a, 2))\}$ forms a perfect matching. $G$ does not have a perfect matching since it has an odd number of vertices.

(c) Take $G$ to be the disjoint union of two triangles, say $G_1$ and $G_2$. $G$ has 6 vertices which is even. From the preceding part one can see that the graph $H$ obtained from the reduction has a perfect matching since we can take the perfect matching in $H_1$ corresponding to $G_1$ and the perfect matching in $H_2$ corresponding to $G_2$ and take their union.
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.

(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 $G$.

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

```
MaxIndSetSize(V,E):
    for k ← 1 to V
        if IndSet(V,E,k+1) = False
            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 $v$.

```
MaxIndSet(G):
    k ← MaxIndSetSize(G)
    for all vertices $v$ of $G$
        if $\text{MaxIndSetSize}(G - v) = k$
            $G ← G - v$
    return $G$
```

Correctness of this algorithm follows inductively from the following claims:

**Claim 1.** $\text{MaxIndSetSize}(G - v) = k$ 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 \texttt{MaxIndSetSize}(V, E) from part (a) as a black box instead. Let \( G + uv \) denote the graph obtained from \( G \) by adding edge \( uv \).

\[\textbf{MaxIndSet}(G):\]
\[
k ← \text{MaxIndSetSize}(G)
\]
\[
\text{if } k = 1
\]
\[
\quad \text{return any vertex}
\]
\[
\text{for all vertices } u
\]
\[
\quad \text{for all vertices } v
\]
\[
\quad \text{if } u \neq v \text{ and } uv \text{ is not an edge}
\]
\[
\quad \quad \text{if } \text{MaxIndSetSize}(G + uv) = k
\]
\[
\quad \quad \quad G ← G + uv
\]
\[
S ← \emptyset
\]
\[
\text{for all vertices } v
\]
\[
\quad \text{if } \deg(v) < V - 1
\]
\[
\quad \quad \text{add } v \text{ to } S
\]
\[
\text{return } S
\]

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 \( G' \):

**Claim 2.** The maximum independent set in \( G' \) is unique.

**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. \( \square \)

**Claim 3.** Suppose \( k > 1 \). The unique maximum independent set of \( G' \) contains vertex \( v \) if and only if \( \deg(v) < V - 1 \).

**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. \( \blacksquare \)
To think about later:

3. Formally, a **proper coloring** of a graph $G = (V,E)$ is a function $c: V \rightarrow \{1,2,\ldots,k\}$, 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.

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 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. ■