1. The Cook-Levin theorem shows that there is a polynomial time reduction reduction \( A \leq_p 3\text{SAT} \) for any language \( A \in \text{NP} \). Sometimes this reduction can be described more easily than the reduction presented in the Cook-Levin theorem.

Show that there is a polynomial time reduction \( 3\text{COL} \leq_p 3\text{SAT} \), where \( 3\text{COL} \) is the problem of deciding if a graph \( G \) is three-colourable.

**Solution:** Given a graph \( G \), generate three variables \( x_{v,1}, x_{v,2}, x_{v,3} \) for each vertex \( v \in G \). We interpret \( x_{v,i} \) as true iff \( v \) is assigned colour \( i \). Generate the following clauses and connect them by AND:

(a) \( x_{v,1} \lor x_{v,2} \lor x_{v,3} \) to ensure each vertex gets a colour.
(b) \( \overline{x_{v,i}} \lor \overline{x_{v,j}} \) for \( i \neq j \) to ensure that no vertex gets more than one colour.
(c) \( \overline{x_{v_1,i}} \lor \overline{x_{v_2,i}} \) for all edges \( (v_1, v_2) \in E(G) \) for \( i = 1, 2, 3 \) to ensure that no two vertices connected by an edge get the same colour.

The Boolean formula generated encodes the conditions needed for \( G \) to have a proper 3-colouring so it is satisfiable if and only if \( G \) is three-colourable. The reduction runs in polynomial time by counting the number of clauses generated.

2. Let \( k > 3 \) be an integer. Show that the problem of deciding if a graph has a \( k \)-colouring is \( \text{NP} \)-Complete assuming that \( 3\text{COL} \) is \( \text{NP} \)-Complete.

**Solution:** Given a potential \( k \)-colouring of \( G \), it can be checked if it is a valid \( k \)-colouring of \( G \) in polynomial time. Now we show that \( 3\text{COL} \leq_p k\text{COL} \). Given a graph \( G \) with vertices \( V \), generate a graph \( H \) by adding \( k-3 \) new vertices \( V' \) to \( G \) connected as a complete graph, and connecting each of the vertices in \( V \) to each of the vertices in \( V' \). This reduction runs in \( O(n) \) time where \( n \) is the number of vertices in \( G \) and produces a graph \( H \) that is \( k \) colourable iff \( G \) was 3 colourable. This is because the vertices in \( V' \) must be assigned \( k-3 \) different colours since they were connected as a complete graph, and the vertices \( V \) must have different colours than the ones in \( V' \). So if \( H \) is \( k \)-colourable, viewing \( G \) as an induced subgraph of \( H \), \( G \) must be 3-colourable.

3. In class, we stated that a Boolean formula \( \phi \) in conjunctive normal form can be converted into a Boolean formula \( \phi' \) with at most 3 literals per clause that is satisfiable iff \( \phi \) is, thus showing a reduction \( \text{SAT} \leq_p 3\text{SAT} \). One way of this doing is converting breaking up any clause \( l_1 \lor \cdots \lor l_n \) in \( \phi \) with \( n > 3 \) literals into two clauses \( (\overline{z_1} \lor l_2 \lor z_1) \land (\overline{l_1} \lor l_3 \lor \cdots \lor l_n) \), where \( z_1 \) is a new variable and then recursively applying this procedure to the longer clause, until each clause has at most three variables.

Prove that \( \phi \) is satisfiable if and only if \( \phi' \) produced by this algorithm from \( \phi \) is satisfiable.
Solution: It suffices to prove the claim for a single clause $\phi_i$ of $\phi$. Unrolling the recursion, we get that every clause $\phi_i = l_1 \lor \cdots \lor l_n$ with $n > 3$ gets converted into the 3-CNF formula

$$\phi'_i = (l_1 \lor l_2 \lor z_1) \land \bigwedge_{i=1}^{n-4} (\overline{z_i} \lor l_{i+2} \lor z_{i+1}) \land (\overline{z_{n-3}} \lor l_{n-1} \lor l_n).$$

If $\phi_i$ clause is satisfiable, then at least one $l_i$ must be true. If $i = 1, 2$, assigning each new variable False satisfies $\phi'_i$. Otherwise, if $i = n-1, n$, assigning each new variable True satisfies $\phi'_i$. Finally, if $3 \leq i \leq n-2$, assigning $z_1, \ldots, z_{i-2}$ True and $z_{i-1}, \ldots, z_{n-3}$ False satisfies $\phi'_i$.

Conversely suppose $\phi'_i$ is satisfiable. Then $\phi_i$ must be satisfiable, since if not, we can conclude that $z_1, z_2, \ldots, z_{n-3}$ are all assigned True to satisfy the first $n-3$ clauses in $\phi'_i$. But this assignment makes the last clause $(\overline{z_{n-3}} \lor l_{n-1} \lor l_n)$ false, which contradicts the assumption that $\phi'_i$ is satisfiable. So $\phi'_i$ satisfiable implies that $\phi_i$ must be too.

4. Recall in the proof of the Cook-Levin theorem we used 2x3 sized windows to check if each row in the computational tableau follows from the previous one according to the transition function of a non-deterministic Turing machine $N$.

Show that the proof would have failed if we tried to use 2x2 sized windows by giving an example of an illegal window that is consistent with the transition function of $N$ across its 2x2 subwindows.

Solution: Consider a non-deterministic Turing machine $N$ with transition function $\delta(q_0, 0) = \{(q_1, 1, L), (q_1, 0, R)\}$ and consider the window

\[
\begin{array}{c|c}
0 & q_0 & 0 \\
\hline
q_1 & 0 & q_1
\end{array}
\]

Clearly the window is illegal since the second row contains two state symbols. However, it is consistent across its 2x2 subwindows since

\[
\begin{array}{c|c}
0 & q_0 & 0 \\
\hline
q_1 & 0 & q_1
\end{array}
\]

and

\[
\begin{array}{c|c}
0 & q_0 & 0 \\
\hline
0 & q_1 & q_1
\end{array}
\]

could have been part of a left move for $N$ and

\[
\begin{array}{c|c}
0 & q_0 & 0 \\
\hline
q_1 & 0 & q_1
\end{array}
\]

could have been part of a right move for $N$.

5. An interesting property of NP-Complete problems is that their search and decision versions are polynomially equivalent, while this is not believed to be true for an arbitrary problem in NP.

Prove that for the case of VERTEX – COVER by showing that if you have an algorithm $A$ that checks if a graph has a vertex cover of size $k$ and only outputs a yes or no answer, then you use $A$ to find a vertex covering of $G$ of size $k$ if one exists.

Solution: Assume that graph $G$ has a vertex covering of size at most $k$. We define an algorithm $FindVC(G, k)$ to determine which vertices cover the edges of $G$ using $A$ as follows:

(a) If $k = 1$, we can output cover $C$ by brute force in polynomial time.

(b) Otherwise, if $k > 1$, let $G_v$ be the graph formed by removing $v$ and all edges adjacent to $v$ from $G$, and run $A(G_v, k-1)$ for each vertex. At least one of these calls must return yes by assumption that $G$ has a cover of size $k$. Once we have found $v$ where $A(G_v, k-1)$ returns a yes answer, add $v$ to a cover $C$ and run $FindVC(G_v, k-1)$ to compute the rest of the cover.