Exercise 2:

Show that CSAT remains NP-complete even if it is restricted to instances in which each variable appears at most three times. Let's call this variant 3VAR-CSAT.

Solution: We show how to reduce CSAT to 3VAR-CSAT.

Given a formula $F$ in CNF, we construct a formula $E$ in CNF where each variable appears at most three times and such that $E$ is satisfiable iff $F$ is satisfiable.

Let $x$ be a variable appearing in $F$ $k$ times, with $k \geq 3$.

Then we construct from $F$ a formula $F_x$ as follows:

1) We replace the $i$-th occurrence of $x_i$ in $F$ with $x_i'$, for $i \in [1, \ldots, k]$, where each $x_i'$ is a fresh variable.

2) We add the following clauses, ensuring that all variables $x_1, \ldots, x_k$ are assigned the same truth value:

$$\overline{x_1} \lor \overline{x_2} \lor (x_2 \lor x_3) \lor \cdots \lor (x_{k-1} \lor x_k) \lor (\overline{x_k} \lor x_1)$$

We have that $F_x$ can be constructed from $F$ in polynomial time, and $F_x$ is satisfiable iff $F$ is satisfiable.

Then, the formula $E$ is obtained from $F$ by repeating the above transformation for each variable $x$ occurring in $F$ more than three times.

We have that:

1) Each variable appears in $E$ at most three times.
2) $E$ can be constructed from $F$ in polynomial time.
3) $E$ is satisfiable iff $F$ is satisfiable.
Exercise 3

Consider 2VARCSAT, i.e. the variant of CSAT in which each variable appears at most two times.

What is the complexity of 2VARCSAT?

Solution: 2VARCSAT is in P.

Let F be an instance of 2VARCSAT.

Notice that for each variable \( x \), we have one of 3 cases:

1) \( x \) appears only positively in \( F \) (one or two times)
2) \( x \) appears only negatively in \( F \) (two times)
3) \( x \) appears one time positively and one time negatively in \( F \)

From this, we obtain the following algorithm to decide the satisfiability of \( F \):

**Input:** set \( F \) of clauses over variables \( x_1, \ldots, x_n \), with each \( x_i \) appearing in at most two clauses

**Output:** YES, if \( F \) is satisfiable, NO otherwise

For each variable \( X \in \{ x_1, \ldots, x_n \} \):

- If \( X \) appears only positively in \( F \) (case 1), then remove from \( F \) the clause(s) in which \( x \) appears
- Else if \( X \) appears only negatively in \( F \) (case 2), then remove from \( F \) the clause(s) in which \( x \) appears
- Else let \( C = x \lor C_{\text{rest}} \) and \( C' = \overline{x} \lor C'_{\text{rest}} \), let the two clauses of \( F \) in which \( x \) appears
  - If \( C_{\text{rest}} \) and \( C'_{\text{rest}} \) are both empty, then answer NO
  - Else remove from \( F \) both \( C \) and \( C' \), and replace them with the clause \( C_{\text{rest}} \lor C'_{\text{rest}} \)

Answer YES
The algorithm runs in polynomial time, since the for-loop is executed $n$ times, and each iteration is at most linear.

Note that a variable $x_i$ might be removed from $F$ before it is considered in the $i$-th iteration of the for-loop. In this case, $F$ is not changed in that iteration.

Moreover:

- In cases (1) and (2), the clauses containing $x_i$ can be trivially satisfied by making $x_i$ true/false.
- In case (3), the algorithm applies a resolution step to $x_i$, and replaces the clauses $C$ and $C'$ with their resolvent.
- By applying a resolution step to a variable $x_{i+1}$ for another variable $x_j$ that has not yet been considered (i.e., $j > i$) and that occurred positively and negatively in two different clauses, the two occurrences of $x_j$ might be merged into a single clause $C_{rest} \lor C'_{rest}$.

This clause can be removed, since it is trivially satisfied by every truth assignment.
Exercise: 2SAT is in P

Idea: we show that 2SAT can be encoded as a graph reachability problem, and then use an algorithm for graph reachability.

1) Encoding of 2SAT as a directed graph reachability problem

Let \( \phi \) be an instance of 2SAT. We define a graph \( G(\phi) \) as follows:
- one node for each variable and for each negated variable
- for each clause \( \alpha \lor \beta \) two edges \( \overline{\alpha} \rightarrow \beta \) \( \overline{\beta} \rightarrow \alpha \)

(note: \( \alpha \lor \beta = \overline{\overline{\alpha} \rightarrow \beta} = \overline{\overline{\beta} \rightarrow \alpha} \))

Example:
- \((x_1 + x_2)\)
- \((x_1 + \overline{x}_3)\)
- \((\overline{x}_1 + x_2)\)
- \((x_2 + \overline{x}_3)\)

Then \( \phi \) is unsatisfiable if there is a variable \( x \) such that \( G(\phi) \) contains two paths \( x \rightarrow \cdots \rightarrow \overline{x} \)

\( \Leftarrow \) Suppose that \( \phi \) has a satisfying truth assignment \( T \).
Assume that \( T(x) = \text{true} \) (a similar argument holds for \( T(x) = \text{false} \)).

Since \( T(x) = \text{true} \) and \( T(\overline{x}) = \text{false} \), and there is a path \( x \rightarrow \cdots \rightarrow \overline{x} \), there must be an edge \( \alpha \rightarrow \beta \) along this path with \( T(\alpha) = \text{true} \) and \( T(\beta) = \text{false} \).
However, since \( X \rightarrow \beta \) is an edge of \( G(\Phi) \), it follows that \( T + \beta \) is a clause of \( \Phi \). This clause is not satisfied by \( T \), which is a contradiction.

\[ \Rightarrow \]

Set \( G(\Phi) \) be a graph that does not contain any node \( \alpha \) with \( \alpha \rightarrow \cdots \rightarrow \beta \).

We construct from such a graph \( G \) a satisfying truth assignment \( T \) by repeating the following step as often as possible:

1. Choose a node \( \alpha \) such that:
   - \( T(\alpha) \) is not yet defined, and
   - there is no path \( \alpha \rightarrow \cdots \rightarrow \beta \).

For every node \( \beta \) that is reachable from \( \alpha \),

1. set \( T(\beta) = \text{true} \) (Note: 2) means to assign false to all predecessors of \( \beta \)).
   
2. set \( T(\beta) = \text{false} \)

Observe (1) the truth assignment \( T \) is well defined, i.e.,

we never have both \( T(\beta) = \text{true} \) and \( T(\beta) = \text{false} \) or \( T(\beta) = \text{false} \) and \( T(\beta) = \text{false} \)

(\( T \) would not be well defined, if we had both \( \alpha \rightarrow \cdots \rightarrow \beta \) and \( \alpha \rightarrow \cdots \rightarrow \beta \)

for some \( \beta \)).

But this cannot happen, since we would have

\[ \alpha \rightarrow \cdots \rightarrow \beta \]

Hence, we would have \( \alpha \rightarrow \cdots \rightarrow \beta \)

2) We assign to all nodes a truth value, since there is no path \( \alpha \rightarrow \cdots \rightarrow \beta \).

3) The truth assignment satisfies all clauses of \( \Phi \), since each clause corresponds to an implication, and there is no \( \alpha \rightarrow \cdots \rightarrow \beta \).