Exercise VIII, Theory of Computation 2025

These exercises are for your own benefit. Feel free to collaborate and share your answers with other students. Solve as many problems as you can and ask for help if you get stuck for too long. Problems marked * are more difficult but also more fun :).

These problems are taken from various sources at EPFL and on the Internet, too numerous to cite individually.

1 Prove that the following problems are in **NP**:

1a A Boolean formula is said to be in *disjunctive normal form (DNF)* if it is an OR ($\vee$) of a number of terms, where each term is an AND ($\wedge$) of some literals. For instance, the following is a DNF formula: $(x \wedge y \wedge \bar{z}) \vee (\bar{y} \wedge z) \vee (\bar{x} \wedge \bar{y})$.

Given a DNF formula, decide it is satisfiable.

1b Given n positive integers $a_1, a_2, \ldots, a_n$, decide if there is some $S \subseteq \{1, 2, \ldots, n\}$ with

$$\sum_{i \in S} a_i = \sum_{i \notin S} a_i.$$

Solution: Recall that to prove a problem/language is in **NP**, we need to provide a polynomial time algorithm, the verifier V, such that given the instance and the witness, V determines if the instance is in the language.

1a For this problem, an instance is a DNF formula F, and as a witness we can take an assignment A of boolean values for the variables. Our verifier V then does the following:

```
On input $(F, A)$, do:
  1. For each term in $F$:
     (a) Check if all literals in the term are true under assignment $A$.
     (b) If yes, accept.
  2. Reject.
```

Note that V runs in linear time in the size of F, A. Moreover, we can see that if $F \in$ DNF-SAT, there exists some A such that $V(F, A)$ accepts. Alternatively, if $F \notin$ DNF-SAT, then for all A, $V(F, A)$ rejects.

1b In this problem, the instance consists of the positive integers $a_1, \ldots, a_n$, and as a witness we take some set $S \subseteq [n]$. Our verifier V then does the following:

```
On input $((a_1, \ldots, a_n), S)$, do:
  1. Compute $s_1 = \sum_{i \in S} a_i$.
  2. Compute $s_2 = \sum_{i \notin S} a_i$.
  3. If $s_1 = s_2$, accept. Otherwise, reject.
```

Similarly, we can see that computing the sums and check if they are equal takes $O(n \log(\max_i a_i))$ time, which is polynomial in terms of the input size. Moreover, we can see that there exists some S such that $V((a_1, \ldots, a_n), S)$ accepts if and only if $(a_1, \ldots, a_n)$ is an accepting instance to the problem.

2 Show that if $L_1, L_2 \subseteq \Sigma^*$ are in **NP** then their concatenation $L = L_1L_2$ is also in **NP**.

Solution: Assume L_1 and L_2 are in **NP**. Then there exists polynomial-time verifiers V_1 and V_2 for deciding membership of L_1 and L_2. An instance to our problem is some string x, and as a witness we take a triple (ℓ, w_1, w_2) where $\ell \geq 0$ is an integer and w_1, w_2 are witnesses for V_1 and V_2 respectively. Now we construct the verifier V for the concatenation.

```
On input $(x, (\ell, w_1, w_2))$, do:
  1. If $\ell > |x|$, reject.
  2. Let $x_1 = x_1 x_2 \cdots x_\ell$.
  3. Let $x_2 = x_{\ell+1} x_{\ell+2} \cdots x_{|x|}$.
  4. Run $V_1(x_1, w_1)$ and $V_2(x_2, w_2)$.
  5. If both $V_1$ and $V_2$ accept, accept.
  6. Otherwise, reject.
```

We first observe that V runs in polynomial-time in terms of the input. Now we show the correctness of V.

- Assume that $x \in L_1L_2$: Then $x = x_1x_2$ for some x_1, x_2 such that $x_1 \in L_1$ and $x_2 \in L_2$. Since L_1 and L_2 are in **NP**, we know that there exist w_1, w_2 such that $V_1(x_1, w_1)$ and $V_2(x_2, w_2)$ accept. Let $w := (|x_1|, w_1, w_2)$, we have that $V(x, w)$ accepts.
- Assume that $x \notin L_1L_2$: For all $0 \leq \ell \leq |x|$, we have that either $x_1 \notin L_1$ or $x_2 \notin L_2$. Therefore, for all w, $V(x, w)$ rejects by construction.

3 Prove that the following problem is **NP**-complete: Given an undirected graph $G = (V, E)$ and a positive integer k, decide if there is a subset S of V with $|S| \geq k$ such that there is an edge between every pair of vertices in S?

Solution: To show a problem is **NP**-complete, we need to first show that the problem is in **NP**, and then show that it is **NP**-hard.

We construct a V verifier for the problem. The instance is the graph G and the integer k, and as the witness we take the set S. Now define V as follows.

```
On input $((V, E, k), S)$, do:
  1. If $|S| < k$, reject.
  2. If $S$ is not a subset of $V$, reject.
  3. If for any $v \neq v' \in S$ we have $(v, v') \notin E$, reject.
  4. Accept.
```

One can easily see that V runs in polynomial time. Moreover, we observe that there exists some S such that $V((G, k), S)$ accepts if and only if (G, k) is an accepting instance of the problem.

Now we show that this problem is **NP**-hard. We know from lecture that determining whether a graph contains an independent set (IS) of size at least k is **NP**-hard. Therefore, if we can provide a polynomial-time reduction from IS to our problem, which we call Clique, then Clique is also **NP**-hard. We construct the function f for the reduction by the following algorithm.

On input (V, E, k), do:

```
1. Construct $E' := \{(u,v) : u \neq v \in V,\ (u,v) \notin E\}$.
2. Output $(V, E', k)$.
```

Clearly, f is polynomial-time computable. It remains to show that $(V, E, k) \in \mathsf{IS}$ if and only if $f(V, E, k) \in \mathsf{Clique}$.

- Assume that $(G, k) \in \mathsf{IS}$: Then there exists a set $S \subseteq V$ such that $|S| \geq k$ and for all $v \neq v' \in S$, $(v, v') \notin E$. By the construction of E', we know that for all $v \neq v \in S$, $(v, v') \in E'$. Therefore, $(V, E', k) \in \mathsf{Clique}$.
- Assume that $f(V, E, k) = (V, E', k) \in \mathsf{Clique}$: Then there exists a set $S \subseteq V$ such that $|S| \geq k$ and for all $v \neq v' \in S$, $(v, v') \in E'$. Therefore, $(v, v') \notin E$ by construction and $(V, E, k) \in \mathsf{IS}$.

Therefore, we can conclude that Clique is **NP**-complete.

4* Let 3SAT3 be the problem of deciding whether a 3CNF formula φ, with the additional assumption that every variable occurs at most 3 times in φ, is satisfiable. Show that 3SAT3 is **NP**-complete.

Solution: We first not that we can show 3SAT3 $\in$ **NP** the same way we use to show SAT $\in$ **NP**. It now remains to show that 3SAT3 is **NP**-hard, we do so by reducing 3SAT to 3SAT3. The function f of the reduction is five by the following algorithm.

```
On input $\varphi$, do:
  1. Let $\varphi' = \varphi$.
  2. For each variable $x$ in $\varphi'$ that appears more than 3 times:
     (a) Let $m$ be the number of occurrences of $x$.
     (b) Introduce $m$ new variables $y_1, \ldots, y_m$.
     (c) Replace the $i$-th occurrence of $x$ in $\varphi'$ by $y_i$.
     (d) Add additional clauses $(y_i \vee \overline{y_{i+1}})$ for all $i \in [m-1]$ and $(y_m \vee \overline{y_1})$ to $\varphi'$.
  3. Output $\varphi'$.
```

We can easily see that f runs in polynomial-time. Now we argue that $\varphi \in \mathsf{3SAT}$ if and only if $f(\varphi) \in \mathsf{3SAT3}$. Note that the conjunction of the clauses $(y_i \vee \overline{y_{i+1}})$ for all $i \in [m-1]$ and $(y_m \vee \overline{y_1})$ is equivalent to saying $y_1 = \cdots = y_m$. Thus, for each such family of variables, setting their value to be the same as the value of the variable x they replace gives us a direct correspondence between satisfying assignments of φ' and those of φ. Moreover, in φ' each y_i appears exactly 3 times. Therefore, by construction, $\varphi \in \mathsf{3SAT} \iff \varphi' \in \mathsf{3SAT3}$.