Chapter 1

NP Completeness

Exercise 1.1 Show that an algorithm that makes at most a constant number of calls to polynomial-time subroutines runs in polynomial time, but that a polynomial number of calls to polynomial-time subroutines may result in an exponential-time algorithm.

Answer: Suppose without loss of generality that algorithm A consists of a sequence of calls to subroutines $S_1, \ldots, S_m$, with each subroutine called once in that order. Assume that each subroutine S_i has a (polynomial) running time bounded by $p_i(n)$, with $p_i(n) \leq p(n) = n^k$. Note that A might call S_1 on its input, then call S_2 on the return value provided by S_1, and so on until S_m is called on the value provided by S_{m-1}. We show by induction that the largest size of the return value and the worst-case running time of the i-th call are both $O(p^i(n))$, with

$$p^i(n) = \overbrace{p(p(\ldots(p(}^{i \text{ times}}n))\ldots) = n^{k^i}.$$

For $i = 1$, the argument of S_1 is of size at most n. Since S_1 has running time $O(p(n))$, its return value has also size $O(p^1(n)) = O(n^k)$. Assume that the proposition holds for any $i < m$, and consider the $(i+1)$-th call. By the inductive hypothesis, the size of the argument of S_{i+1} has size $O(p^i(n))$. Since S_{i+1} has running time $O(p(n))$, the running time of the $(i+1)$-th call and the size of the return value are both $O\left((p^i(n))^k\right) = O(p^{i+1}(n))$. The inductive thesis follows.

After the m-th call, we have taken time

$$O\left(\sum_{i=1}^{m} p^i(n)\right) = O\left(mp^m(n)\right) = O\left(mn^{k^m}\right),$$

which is polynomial for any constant k and m.

On the other hand, suppose that A simply makes n nested calls to a subroutine S, i.e., on input n, A computes

$$S^n(n) = \overbrace{S(S(\ldots(S(}^{n \text{ times}}n))\ldots).$$

Suppose that S takes linear time and that its return value is twice as long as its input. It follows that the running time and the size of the return value of the i-th call are both $\Theta(n2^i)$. Therefore, the total running time is

$$\Theta\left(n\sum_{i=1}^{n} 2^i\right) = \Theta(n2^n).$$

$\square$

Exercise 1.2 Prove that the class NP of languages is closed under the following operations:

(a) Union of two languages.

(b) Intersection of two languages.

(c) Concatenation of two languages.

(d) Kleene star of a language.

Answer: Observe that we can re-state the definition of $L \in NP$ as follows:
Definition A language L is in NP iff there exists a verification algorithm A, and polynomials p, q such that:

- $L = L_A$;
- $\forall x \in L, \exists y$ such that $|y| \leq p(|x|)$ and $A(x, y) = 1$;
- A on input (x, y) halts in time $\leq q(|x| + |y|)$.

In what follows we use the notation $(p+q)(n)$ to denote the polynomial whose value on n is $p(n) + q(n)$.

(a) Let $L_1, L_2 \in NP$, with verification algorithms A_1, A_2 (i.e., $L_1 = L_{A_1}$, $L_2 = L_{A_2}$), and polynomial bounds p_1, q_1, and p_2, q_2, respectively.

Define a new verification algorithm A as follows:

$A(x,y)$
if $A_1(x,y) = 1$
  then return 1
  else return $A_2(x,y)$

Note that $A(x,y) = 1$ iff $A_1(x,y) = 1$ or $A_2(x,y) = 1$. We have:

1. $L_1 \cup L_2 \subseteq L_A$. Let $x \in L_1 \cup L_2$. Then $x \in L_1$ or $x \in L_2$. If $x \in L_1$, then $\exists y$ such that $A_1(x,y) = 1$. Hence, $A(x,y) = 1$. Otherwise, if $x \in L_2$, then $\exists y$ such that $A_2(x,y) = 1$. Hence, $A(x,y) = 1$. Therefore $x \in L_A$.

2. $L_A \subseteq L_1 \cup L_2$. Let $x \in L_A$. Then $\exists y$ such that $A(x,y) = 1$. This implies that either $A_1(x,y) = 1$ or $A_2(x,y) = 1$, that is, $x \in L_{A_1}$ or $x \in L_{A_2}$. Therefore $x \in L_{A_1} \cup L_{A_2} = L_1 \cup L_2$.

3. $\forall x \in L_A, \exists y$ such that $A(x,y) = 1$. If $x \in L_1$, we have $|y| \leq p_1(|x|)$. If $x \in L_2$, we have $|y| \leq p_2(|x|)$. Threfore $|y| \leq p_1(|x|) + p_2(|x|) = (p_1 + p_2)(|x|)$.

4. A on (x,y) takes time $O((q_1 + q_2)(|x| + |y|))$ and is therefore polynomially bounded.

This proves that $L_1 \cup L_2 \in NP$.

(b) Let $L_1, L_2 \in NP$, with verification algorithms A_1, A_2, and polynomial bounds p_1, q_1 and p_2, q_2, respectively. Moreover, let $\#$ be a distinguished character not in the alphabet of the certificates. Define a new verification algorithm A as follows:

$A(x,y)$
if $y \neq y_1\#y_2$
  then return 0
if $A_1(x,y_1) = 1$
  then if $A_2(x,y_2) = 1$
      then return 1
return 0

Note that $A(x,y) = 1$ iff $y = y_1\#y_2$ and $A_1(x,y_1) = A_2(x,y_2) = 1$. We have:

1. $L_1 \cap L_2 \subseteq L_A$. Let $x \in L_1 \cap L_2$. Then $x \in L_1$ and $x \in L_2$. Then, $\exists y_1, y_2$ such that $A_1(x,y_1) = 1$ and $A_2(x,y_2) = 1$. This implies that $A(x, y_1\#y_2) = 1$. Therefore $x \in L_A$.

2. $L_A \subseteq L_1 \cap L_2$. Let $x \in L_A$. Then $\exists y_1\#y_2$ such that $A(x, y_1\#y_2) = 1$. This implies that $A_1(x,y_1) = 1$ and $A_2(x,y_2) = 1$. Hence $x \in L_{A_1}$ and $x \in L_{A_2}$. Therefore, $x \in L_1 \cap L_2$.

3. $\forall x \in L_A, \exists y$ such that $A(x,y) = 1$. Moreover, since $y = y_1\#y_2$, with $|y_1| \le p_1(|x|)$ and $|y_2| \le p_2(|x|)$, we have $|y| = |y_1| + |y_2| + 1 \le (p_1 + p_2)(|x|) + 1$. Therefore $|y|$ is polynomially bounded.

4. A on (x, y) runs in time $O((q_1 + q_2)(|x| + |y|))$.

This proves that $L_1 \cap L_2 \in NP$.

(c) Given a string x, let $x_{i\ldots j}$ denote the substring of x (of length $j - i + 1$) from the ith to the jth character. Define $x_{i\ldots j} = \varepsilon$ if $i > j$. Let $L_1, L_2 \in NP$, with verification algorithms A_1, A_2, and polynomial bounds p_1, q_1 and p_2, q_2, respectively. Moreover, let # be a distinguished character not in the alphabet of the certificates. Define a new verification algorithm A as follows:

$A(x, y)$
if $y \neq y_1\#y_2$
  then return 0
for $k \leftarrow 0$ **to** $|x|$**do**
    if $A_1(x_{1\ldots k}, y_1) = 1$ **and** $A_2(x_{k+1\ldots|x|}, y_2) = 1$
      then return 1
return 0

Note that $A(x,y) = 1$ iff $y = y_1\#y_2$ and $\exists 0 \le k \le |x|$ such that $A_1(x_{1\ldots k}, y_1) = 1$ and $A_2(x_{k+1\ldots|x|}, y_2) = 1$. We have:

1. $L_1L_2 \subseteq L_A$. Let $x \in L_1L_2$. Then $\exists 0 \le k \le |x|$ such that $x_{1\ldots k} \in L_1$ and $x_{k+1\ldots|x|} \in L_2$. Hence, $\exists y_1, y_2$ such that $A_1(x_{1\ldots k}, y_1) = 1$ and $A_2(x_{k+1\ldots|x|}, y_2) = 1$. So, $A(x, y_1\#y_2) = 1$, i.e. $x \in L_A$.

2. $L_A \subseteq L_1L_2$. This is immediate from our definition of A.

3. $\forall x \in L_A$, $\exists y$ such that $A(x,y) = 1$ and $|y| \le (p_1 + p_2)(|x|) + 1$.

4. When running A on (x, y), there are at most $|x|+1$ executions of A_1, each taking time $\le q_1(|x| + |y|)$, and at most $|x| + 1$ executions of A_2, each taking time $\le q_2(|x| + |y|)$. So, A has a polynomial time bound $O(|x|(q_1 + q_2)(|x| + |y|))$.

This proves that $L_1L_2 \in NP$.

(d) We can exploit the advantage of guessing the right certificate by encoding the substring divisions of x in the certificate y. Namely, let #, & be distinguished characters not in the alphabet of the certificates. A certificate for a string x in $L^\star$ will be of type

$$y = y_1\#y_2\#\ldots\#y_k\#m_1\&m_2\&\ldots\&m_{k-1},$$

where $1 \leq k \leq |x|$, $m_0 = 0 \leq m_1 \leq \ldots m_{k-1} \leq m_k = |x|$, and, for any i, $1 \leq i \leq k$, y_i is a potential certificate for $x_{m_{i-1}+1\ldots m_i}$'s membership in L. Define a new verification algorithm A as follows:

$A(x, y)$
for $k \leftarrow 1$ **to** $|x|$**do**
  $m_0 \leftarrow 0, m_k \leftarrow |x|$
  if $y = y_1\#y_2\#\ldots\#y_k\#m_1\&m_2\&\ldots\&m_{k-1}$
   then $t \leftarrow$ **true**
    for $i \leftarrow 1$ **to** k **do**
     do $t \leftarrow t$ **and** $A_0(x_{m_{i-1}+1\ldots m_i}, y_i)$
    if t **then return** 1
return 0

$A(x, y) = 1$ iff $\exists k, 1 \leq k \leq |x|$, such that $y = y_1\#y_2\#\ldots\#y_k\#m_1\&m_2\&\ldots\&m_{k-1}$ and, for any i, $1 \leq i \leq k$, $A_0(x_{m_{i-1}\ldots m_i}, y_i) = 1$. We have:

1. $L^\star \subseteq L_A$. Let $x \in L^\star$. Then there is a value k, $1 \leq k \leq |x|$, such that x is the concatenation of strings $x_{m_{i-1}+1\ldots m_i} \in L$, for $1 \leq i \leq k$. Then, for each such i there is a y_i such that $A_0(x_{m_{i-1}\ldots m_i}, y_i) = 1$. Thus, if $y = y_1\#y_2\#\ldots\#y_k\#m_1\&m_2\&\cdots\&m_{k-1}$, we have $A(x, y) = 1$. Therefore, $x \in L_A$.

2. $L_A \subseteq L^\star$. Let $x \in L_A$. Then, there is a $y = y_1\#y_2\#\cdots\#y_k\#m_1\&m_2\&\ldots\&m_{k-1}$ such that $A(x, y) = 1$. By our definition of A, this implies that $x_{m_{i-1}\ldots m_i} \in L$ for any i, $1 \leq i \leq k$. Therefore, $x \in L^\star$.

3. Since there are at most $|x|$ y_i's, with $|y_i| \leq p_0(|x|)$, and at most $|x|$ m_i's, with $|m_i| \leq \log|x|$, and at most $2|x|$ extra-characters in y, we have $|y| = O(|x|(p_0(|x|) + \log|x| + 2))$, which is polynomially bounded.

4. A on (x, y) runs A_0 at most $|x|$ times (because $k \leq |x|$), each taking time $\leq q_0(|x| + |y|)$. Thus, A runs in time $O(|x|q_0(|x| + |y|))$, and is therefore polynomially bounded.

This proves that $L^\star \in NP$. □

Exercise 1.3 Prove that $<_P$ is a transitive relation. That is, for $L_1, L_2, L_3 \subseteq \{0,1\}^\star$,

$$(L_1 <_P L_2 \;\text{ and }\; L_2 <_P L_3) \Rightarrow L_1 <_P L_3.$$

Answer: Let $f(x), g(x)$ denote the polynomial-time computable functions that reduce L_1 to L_2 and L_2 to L_3, respectively. Let $h(x) = g(f(x))$. For all strings $x \in \{0,1\}^\star$ we have:

$$\begin{aligned} x \in L_1 \quad &\text{iff} \quad f(x) \in L_2 \\ y = f(x) \in L_2 \quad &\text{iff} \quad g(y) = g(f(x)) \in L_3 \end{aligned}$$

Hence

$$x \in L_1 \quad \text{iff} \quad h(x) = g(f(x)) \in L_3.$$

Note that $h(x) = g(f(x))$ is polynomial-time computable, since it is the composition of two polynomial-time computable functions. This proves that $L_1 <_P L_3$. □

Exercise 1.4 We say that a function f is computable in *quasi linear* time $T_f(n)$ if there are nonnegative constants c and k such that $T_f(n) \leq cn(\log n)^k$. Show that reducibility in quasi linear time is a transitive relation.

Answer: Consider three languages L_1, L_2 and L_3 such that L_1 is reducible in quasi linear time to L_2, and L_2 is reducible in quasi linear time to L_3. By the definition of reduction, there exist reduction functions f from L_1 to L_2 computable in quasi linear time $T_f(n) \leq c_f n(\log n)^{k_f}$, and g from L_2 to L_3 computable in quasi linear time $T_g(n) \leq c_g n(\log n)^{k_g}$. In the previous exercise, we have shown that $h(x) = g(f(x))$ is a reduction function from L_1 to L_3. It remains to show that $h(x)$ is computable in quasi linear time.

Let $y = f(x)$ and $h(x) = g(y)$. Let also $|x| = n$. We have $|y| \leq T_f(|x|) \leq c_f n(\log n)^{k_f}$. Therefore, $h(x) = g(y)$ can be computed in time

$$\begin{aligned} T_h(n) &\leq T_f(n) + T_g(T_f(n)) \\ &= c_f n(\log n)^{k_f} + c_g \left(c_f n(\log n)^{k_f}\right) \left(\log \left(c_f n(\log n)^{k_f}\right)\right)^{k_g} \\ &= (c_g c_f) n \left(\log n\right)^{k_f + k_g} (1 + o(1)) \end{aligned}$$

Therefore, there exist constants $c_h > c_g c_f$ and $k_h = k_f + k_g$ such that $T_h(n) \leq c_h n(\log n)^{k_h}$. This shows that L_1 is reducible in quasi linear time to L_3. □

Exercise 1.5 Prove that $L \leq_P L^c$ iff $L^c \leq_P L$.

Answer:

$$
\begin{aligned}
f \text{ reduces } L \text{ to } L^c \quad & \Leftrightarrow \quad \forall x \in \Sigma^\star : \ x \in L \text{ iff } f(x) \in L^c \\
& \Leftrightarrow \quad \forall x \in \Sigma^\star : \ (x \notin L) \text{ iff } (f(x) \notin L^c) \\
& \Leftrightarrow \quad \forall x \in \Sigma^\star : \ x \in L^c \text{ iff } f(x) \in L \\
& \Leftrightarrow \quad f \text{ reduces } L^c \text{ to } L.
\end{aligned}
$$

□

Exercise 1.6 Under the assumption that $P \neq NP$, prove or disprove the following statements:

(a) $\{0,1\}^\star \in P$.

(b) There are NP-complete languages that are regular. Recall that a regular language is one which is accepted by a Deterministic Finite-State Automaton (DFSA).

(c) If L contains an NP-complete subset, then L is NP-complete.

(d) All NP-Complete problems can be solved in time $O\left(2^{p(n)}\right)$, for some polynomial $p(n)$.

(e) The *halting problem* is NP-complete.

(f) The *halting problem* is NP-hard.

Answer:

(a) True $\{0,1\}^\star$ is decided by the following constant-time algorithm:

$A_{\{0,1\}^\star}(x)$
return 1

(b) False Given a regular language L, any DFSA that accepts L yields a linear-time decision algorithm A_L for L. To see this, associate a distinct label to each state and use conditional jumps to "simulate" transitions. On string x, we will perform exactly $|x|$ jumps before either accepting or rejecting, according to whether the last jump leads to a final or a nonfinal state. This proves that for any regular language L, $L \in P$.

(c) False Counterexample: $\{0,1\}^\star \supset L_{\mathrm{SAT}}$, but Point **(a)** proves that $\{0,1\}^\star \in P$.

(d) True For $L \in NP$, let A_L be the polynomial-time algorithm verifying L and running in time $T_A(|x|+|y|) \leq c_1(|x|+|y|)^h$, where $|y| \leq c_2|x|^k$ when $x \in L$. We can write the following decision algorithm for L:

DECIDE_$L(x)$
for each $y \in \{0,1\}^\star$, $|y| \leq c_2|x|^k$ **do**
  if $A_L(x,y) = 1$ **then return** 1
return 0

DECIDE_$L(x)$ returns 1 if and only if there exists a "short" certificate for x, which is the case if and only if $x \in L$. Therefore DECIDE_L decides L. The running time of DECIDE_$L(x)$ is $O\left(|x|^{hk}2^{c_2|x|^k}\right) = O\left(2^{c_2|x|^k+|x|}\right) = O\left(2^{p(|x|)}\right)$.

(e) False Recall that the halting problem corresponds to the following language:

$$L_H = \{y \in \{0,1\}^\star : y = \langle M, x\rangle, M \text{ is a Turing machine which terminates on input } x\}.$$

We know that L_H is an undecidable language. On the other hand, since $NPC \subseteq NP$, Point **(d)** proves that any NP-Complete problem is decidable. Therefore the halting problem cannot be NP-Complete.

(f) True Consider an arbitrary language $L \in NP$, and let DECIDE_L be the exponential decision algorithm for L developed in Point **(d)**. Consider the following program, based on DECIDE_L:

$A_L(x)$
if DECIDE_$L(x) = 1$
  then return 1
  else while true do
    { loop forever }

A_L either returns 1 or goes into an infinite loop. Let M_{A_L} be a Turing Machine encoding algorithm A_L. Define the following function:

$$f(x) = \langle M_{A_L}, x\rangle$$

Clearly, f is computable in polynomial time, since it takes constant time to encode the Turing Machine and linear time to copy the input string. We now prove that f reduces L

to L_H, the language of the halting problem. We have

$$\begin{aligned} x \in L &\Leftrightarrow \text{DECIDE\_}L(x) = 1 \\ &\Leftrightarrow A_L(x) \text{ terminates} \\ &\Leftrightarrow \langle M_{A_L}, x\rangle \in L_H \end{aligned}$$

We have proved that for any language $L \in NP$, $L <_P L_H$. Hence L_H is NP-Hard. □

Exercise 1.7 Suppose that someone gives you a polynomial-time algorithm to decide formula satisfiability. Describe how to use this algorithm to find satisfying assignments in polynomial time.

Answer: Let $\Phi(x_1, ..., x_m)$ be a boolean formula, and let SAT be a (rather unlikely) subroutine deciding satisfiability in polynomial time $O(p(n))$, where $n \geq m$ is the size of formula Φ. We can find a satisfying assignment to Φ (assuming that there is one, which can be ascertained with one call to SAT) by iteratively finding a truth assignment $s(1)$ for x_1, then finding an assignment $s(2)$ for x_2, and so on until we have an assignment for all the variables. Our invariant will be that after the i-th iteration, the formula $\Phi(s(1), \ldots, s(i), x_{i+1}, \ldots, x_m)$ (i.e., the formula where the variables $x_1, \ldots, x_i$ are substituted with the boolean constants $s(1), \ldots, s(i) \in \{\mathbf{false}, \mathbf{true}\}$) is satisfiable.

The algorithm works as follows: having found assignments $s(1), s(2), \ldots, s(i-1)$ for the first $i-1$ variables, we call SAT on $\Phi(s(1), \ldots, s(i-1), \mathbf{false}, x_{i+1}, \ldots, x_m)$. If this formula is satisfiable, then $s(i) = \mathbf{false}$. If the formula is not satisfiable, then $s(i) = \mathbf{true}$. In the latest case, $\Phi(s(1), \ldots, s(i-1), \mathbf{true}, x_{i+1}, \ldots, x_m)$ must be satisfiable, because our loop invariant/induction hypothesis tells us that $\Phi(s(1), \ldots, s(i-1), x_i, \ldots, x_m)$ is satisfiable (and $\Phi(s(1), \ldots, s(i-1), \mathbf{false}, x_{i+1}, \ldots, x_m)$ is not). The algorithm follows:

FIND_ASSIGNMENT($\Phi(x_1, x_2, \ldots, x_m)$)
if SAT($\Phi(x_1, x_2, \ldots, x_m)$)= "no"
  then return "formula is not satisfiable"
for $i \leftarrow 1$ **to** m
  do $s[i] \leftarrow$ **false**
  if SAT($\Phi(s[1], \ldots, s[i], x_{i+1}, \ldots, x_m)$) = "no"
    then $s[i] \leftarrow$ **true**
return s

At stage i, it takes polynomial time to prepare $\Phi(s(1), \ldots, s(i), \mathbf{false}, x_{i+2}, \ldots, x_m)$; then SAT takes time $p(n)$ to decide the satisfiability of this formula. Since there are $m = O(n)$ iterations, the overall running time is polynomial. □

Exercise 1.8 Consider the following decision problem:

BI_SAT (DOUBLE SATISFIABILITY):
INSTANCE: $\langle\Phi(x_1, x_2, \ldots, x_n)\rangle$, Φ is a boolean formula

QUESTION: Are there two *distinct* satisfying assignments for Φ?

Show that BI_SAT is NP-Complete.

Answer: Let us first show that BI_SAT $\in NP$. Consider the following straightforward algorithm.

VERIFY_BI_SAT(x, y)
if $x \neq \langle\Phi(x_1, x_2, \ldots, x_n)\rangle$
  then return 0
if $y \neq \langle(b_1^1, b_2^1, \ldots, b_n^1), (b_1^2, b_2^2, \ldots, b_n^2)\rangle$
  then return 0
{ the b_i^j's are boolean values that form two
  truth assignments for the variables of Φ }
same $\leftarrow$ **true**
for $i \leftarrow 1$ **to** n **do** *same* $\leftarrow$ *same* **and** $(b_i^1 = b_i^2)$
if *same* **then return** 0
{ truth assignments must be distinct}
if $\Phi(b_1^1, b_2^1, \ldots, b_n^1)$ **and** $\Phi(b_1^2, b_2^2, \ldots, b_n^2)$
  then return 1
return 0

The algorithm performs two evaluations of Φ plus some extra steps whose number is linear in $|\langle\Phi\rangle|$. Since a boolean formula can be evaluated in time polynomial in its length, VERIFY_BI_SAT verifies BI_SAT in polynomial time.

The second step is to show that BI_SAT is NP-Hard. We show that SAT $<_P$ BI_SAT, where SAT is the Boolean Formula Satisfiability problem.

Let $\Phi(x_1, x_2, \ldots, x_n)$ be a formula, and let x_{n+1} be a new variable. We define our reduction function as follows:

$$f(\langle\Phi(x_1, x_2, \ldots, x_n)\rangle = \langle\Phi(x_1, x_2, \ldots, x_n) \wedge (x_{n+1} \vee \neg x_{n+1})\rangle.$$

Let us show that

$$\langle\Phi(x_1, x_2, \ldots, x_n)\rangle \in \text{SAT} \Leftrightarrow f(\langle\Phi(x_1, x_2, \ldots, x_n)\rangle) \in \text{BI\_SAT}.$$

Suppose $\Phi(x_1, x_2, \ldots, x_n) \in$ SAT. Then there is a truth assignment $(b_1, b_2, \ldots, b_n)$ to variables $(x_1, x_2, \ldots, x_n)$ satisfying $\Phi(x_1, x_2, \ldots, x_n)$. Since $(x_{n+1} \vee \neg x_{n+1})$ is true for both

$x_{n+1} = \textbf{false}$ and $x_{n+1} = \textbf{true}$, we have that $f(\Phi(x_1, x_2, \ldots, x_n))$ is satisfied by the two assignments $(b_1, b_2, \ldots, b_n, \textbf{false})$ and $(b_1, b_2, \ldots, b_n, \textbf{true})$. Conversely, if $f(\Phi(x_1, x_2, \ldots, x_n))$ has two satisfiying assignments $(b_1^1, \ldots, b_n^1, b_{n+1}^1)$ and $(b_1^2, \ldots, b_n^2, b_{n+1}^2)$ then $\Phi(x_1, x_2, \ldots, x_n)$ is clearly satisfied by both assignments $(b_1^1, b_2^1, \ldots, b_n^1)$ and $(b_1^2, b_2^2, \ldots, b_n^2)$, since, in order for $\Phi(x_1, x_2, \ldots, x_n) \wedge (x_{n+1} \vee \neg x_{n+1})$ to be true, both operands $\Phi(x_1, x_2, \ldots, x_n)$ and $(x_{n+1} \vee \neg x_{n+1})$ must be true.

Finally, note that f creates a new variable x_{n+1} and computes the encoding of the new formula. Such activity can be accomplished in time polynomial in $|\langle \Phi(x_1, x_2, \ldots, x_n) \rangle|$. $\square$

Exercise 1.9 Consider the following decision problem:

> M_SAT (MAJORITY SATISFIABILITY):
> **INSTANCE**: $\langle \Phi(x_1, x_2, \ldots, x_n) \rangle$, Φ is a boolean formula
>
> **QUESTION**: Is $\Phi(x_1, x_2, \ldots, x_n)$ true for *more* than a half of the possible 2^n input assignments?

Show that M_SAT is NP-hard.

Answer: We show that SAT $<_P$ M_SAT. Given a formula $\Phi(x_1, x_2, \ldots, x_n)$, define

$$\begin{aligned} f(\langle \Phi(x_1, x_2, \ldots, x_n) \rangle) \quad &= \quad \langle \Phi'(x_1, x_2, \ldots, x_n, x_{n+1}) \rangle, \\ \text{with} \quad & \Phi'(x_1, x_2, \ldots, x_n, x_{n+1}) = \Phi(x_1, x_2, \ldots, x_n) \vee x_{n+1}. \end{aligned}$$

Note that f is trivially computable in time polynomial in $|\langle \Phi(x_1, x_2, \ldots, x_n) \rangle|$.

Let us show that f reduces SAT to M_SAT. First note that Φ' is satisfied by any of the 2^n assignments $(x_1, x_2, \ldots, x_n, \textbf{true})$. If $\Phi \in$ SAT, then there exists an assignment $(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n)$ such that $\Phi(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n) = \textbf{true}$. Then, Φ' is also satisfied by the assignment $(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n, \textbf{false})$, for a total of at least $2^n + 1 = 2^{n+1}/2 + 1$ satisfying assignments, therefore $f(\langle \Phi \rangle) \in$ M_SAT. Vice versa, if Φ is not satisfiable, then the assignments $(x_1, x_2, \ldots, x_n, \textbf{true})$ are all and only those satisfying Φ'. Since these are $2^n < 2^{n+1}/2 + 1$, $f(\langle \Phi \rangle) \notin$ M_SAT. $\square$

Exercise 1.10 Consider the following decision problem:

> 0-1 IP (0-1 INTEGER PROGRAMMING):
> **INSTANCE**: $\langle A, \boldsymbol{b} \rangle$, where A is an integer $m \times n$ matrix and $\boldsymbol{b}$ is an integer m-vector.

QUESTION: Is there an n-vector $\boldsymbol{x}$ with components in $\{0,1\}$ such that $(A\boldsymbol{x})_i \geq b_i$, for $1 \leq i \leq m$?

Prove that 0-1 IP is NP-complete.

Answer: A certificate for an instance $(A, \boldsymbol{b})$ of 0-1 IP is clearly a 0-1 cols(A)-vector x. Here is the verification algorithm:

VERIFY_IP(a, y)
if $(a \neq \langle A, \boldsymbol{b}\rangle)$ **or** $(y \neq \langle \boldsymbol{x}\rangle)$
  then return 0
$m \leftarrow$ rows(A)
$n \leftarrow$ cols(A)
if (length$(\boldsymbol{b}) \neq m$) **or** (length$(\boldsymbol{x}) \neq n$)
  then return 0
for $i \leftarrow 1$ **to** m **do**
  for $j \leftarrow 1$ **to** n **do**
    if ($a_{i,j}$, b_i noninteger) **or** $(x_j \notin \{0,1\})$
      then return 0
$\boldsymbol{c} \leftarrow$ MAT_VEC_MULT$(A, \boldsymbol{x})$
for $i \leftarrow 1$ **to** m **do**
  if $c_i < b_i$ **then return** 0
return 1

VERIFY_IP(a, y) is a legal verification algorithm for 0-1 IP, since it returns 1 if and only if a is a well-formed encoding $\langle A, \boldsymbol{b}\rangle$ of an instance of IP, y is a well formed encoding of a 0-1 cols(A)-vector $\boldsymbol{x}$, and $A\boldsymbol{x} \geq \boldsymbol{b}$. Moreover, since matrix-vector multiplication can be performed in polynomial time, the algorithm is clearly polynomial.

To show 0-1 IP is NP-hard, we show that 3-CNF-SAT $\leq_P$ 0-1 IP. Let $\Phi(x_1, x_2, \ldots x_n) = C_1 \wedge C_2 \wedge \cdots \wedge C_k$ be a boolean formula in 3-CNF made of k clauses. Without loss of generality, in what follows we assume than no clause C_j contains both x_i and $\overline{x_i}$, since in this case C_j is a tautology and can be eliminated from Φ without affecting the value of the formula on any of the assignments. We will say that $x_i = 1$ if x_i is assigned the value **true**, and $x_i = 0$ if x_i is assigned the value **false**. If a boolean variable has value $x_j = \alpha$, with $\alpha \in \{0,1\}$, then the value of $\overline{x_j}$ is $(1-\alpha)$. With this convention, a 0-1 n-vector can be seen as a truth assignment to the n boolean variables of Φ.

From our instance Φ of 3-CNF-SAT, we build an instance $(A, \boldsymbol{b})$ of 0-1 IP in the following way:

- A is a $k \times n$ matrix, where row i is built from clause C_i of Φ in the following way: if boolean variable x_j does not appear in C_i, then $a_{i,j} = 0$. If x_j is a literal in C_i, then $a_{i,j} = 1$. If $\overline{x_j}$ is a literal in C_i, then $a_{i,j} = -1$.

- $\boldsymbol{b}$ is a k-vector such that $b_i = 1 - |\{\text{negative literals in } C_i\}|$.

Given the above definition of A and $\boldsymbol{b}$, for $1 \le i \le k$, the i-th inequality $(A\boldsymbol{x})_i \ge b_i$ can be rewritten as follows:

$$\sum_{x_j \in C_i} x_j + \sum_{\overline{x_j} \in C_i} (1 - x_j) \ge 1. \tag{1.1}$$

Assume now that Φ is satisfiable. Then there must exist a truth assignment to the n variables such that satisfies all clauses. Let $(t_1, t_2, \ldots, t_n)$ be the 0-1 n-vector corresponding to such assignment. Then, the sum of the values $\alpha_i^1, \alpha_i^2, \alpha_i^3$ of the three literals in each clause C_i dictated by the t_j's is at least 1. Hence, all inequalities are satisfied at the same time by setting $x_j = 1$ if $t_j = \textbf{true}$, and $x_j = 0$ otherwise. Vice versa, any 0-1 n-vector $(x_1, x_2, \ldots, x_n)$ satisfying all the k inequalities yields a satisfying truth assignment for Φ. Hence, f is a reduction from 3-CNF-SAT to IP. □

Exercise 1.11 Consider the following decision problem:

DF (DISTINCT FORMULAE):

INSTANCE: $\langle \Phi(x_1, x_2, \ldots, x_n), \Psi(x_1, x_2, \ldots, x_n) \rangle$, with $\Phi(x_1, x_2, \ldots, x_n)$ and $\Psi(x_1, x_2, \ldots, x_n)$ boolean formulae.

QUESTION: Is there a truth assignment $(b_1, b_2, \ldots, b_n)$ such that $\Phi(b_1, b_2, \ldots, b_n) \neq \Psi(b_1, b_2, \ldots, b_n)$?

Show that DF is NP-complete.

Answer: We first show that DF $\in NP$. A candidate certificate for DF is a truth assignment to the n variables. The verification algorithm VERIFY_DF first checks whether its first input $x = \langle \Phi(x_1, x_2, \ldots, x_n), \Psi(x_1, x_2, \ldots, x_n) \rangle$, that is, x is a well-formed encoding of an instance of DF; then checks that its second input encodes a truth assignment $(b_1, b_2, \ldots, b_n)$. If this is the case, then the algorithm checks whether $\Phi(b_1, b_2, \ldots, b_n) \neq \Psi(b_1, b_2, \ldots, b_n)$. The running time of VERIFY_DF is clearly polynomial in the size of its inputs. For brevity, we omit the code of the algorithm.

In order to show that DF is NP-Hard, we provide a polynomial-time reduction from SAT to DF. Recall that an instance of SAT is $\langle \Phi(x_1, x_2, \ldots, x_n) \rangle$ and the question is whether Φ is satisfiable, that is, whether there is a truth assignment $(b_1, b_2, \ldots, b_n)$ such that $\Phi(b_1, b_2, \ldots, b_n) = \textbf{true}$. Our reduction function is the following:

$$f\left(\langle \Phi(x_1, x_2, \ldots, x_n) \rangle\right) = \langle \Phi(x_1, x_2, \ldots, x_n), \Psi(x_1, x_2, \ldots, x_n) = x_1 \wedge \neg x_1 \rangle.$$

Note that the second formula in $f(\langle\Phi(x_1, x_2, \ldots, x_n)\rangle)$ is a contradiction, therefore its evaluation yields **false** on all truth assignments.

Clearly, f is computable in polynomial time. It remains to show that f is indeed a reduction. Assume that $\langle\Phi(x_1, x_2, \ldots, x_n)\rangle \in$ SAT. Then there is a truth assignment $(b_1, b_2, \ldots, b_n)$ such that $\Phi(b_1, b_2, \ldots, b_n) = \textbf{true}$. On such assignment, we have

$$\textbf{true} = \Phi(b_1, b_2, \ldots, b_n) \neq \Psi(b_1, b_2, \ldots, b_n) = \textbf{false},$$

hence $f(\langle\Phi(x_1, x_2, \ldots, x_n)\rangle) \in$ DF. Vice versa, if $\langle\Phi(x_1, x_2, \ldots, x_n)\rangle \notin$ SAT, then

$$\Phi(b_1, b_2, \ldots, b_n) = \Psi(b_1, b_2, \ldots, b_n) = \textbf{false},$$

on *all* truth assignments $(b_1, b_2, \ldots, b_n)$. Therefore $f(\langle\Phi(x_1, x_2, \ldots, x_n)\rangle) \notin$DF. □

Exercise 1.12 Consider the following problem:

TWO-CLIQUE :
INSTANCE: $\langle G, h, k\rangle$, with G an undirected graph and $h, k > 0$.

QUESTION: Does G contain two *disjoint* cliques of size h and k?

(a) Show that TWO-CLIQUE is in NP.

(b) Show that TWO-CLIQUE is NP-hard.

Answer:

(a) Consider the following verification algorithm A.

$A(x, y)$
if $x \neq \langle G = (V, E), h, k\rangle, h, k > 0$
  then return 0
if $y \neq \langle U_1, U_2\rangle, U_1, U_2 \subset V$
  then return 0
if $|U_1| = h$ **and** $|U_2| = k$ **and** $U_1 \cap U_2 = \emptyset$
  then if IS_CLIQUE(G, U_1) **and** IS_CLIQUE(G, U_2)
    then return 1
return 0

Subroutine IS_CLIQUE(G, U) checks the adjacency list of G to make sure that U is a clique. Clearly, $L_A =$ TWO-CLIQUE. The length of an accepting certificate y is clearly $O(|V|) = O(|x|)$. Finally, IS_CLIQUE(G, U) can clearly be implemented in polynomial time, therefore A is polynomial.

(b) Let us consider the following reduction function f from CLIQUE to TWO-CLIQUE.

$$f(\langle G = (V, E), h \rangle) = \langle G' = (V \cup \{u\}, E), h, 1 \rangle,$$

where $\langle G = (V, E), h \rangle$ is a CLIQUE instance and $u \notin V$. Note that u is an isolated node in G'.

Let us first prove that f is indeed a reduction. If $\langle G = (V, E), h \rangle \in$ CLIQUE then there is a subset K of V which forms an h-clique. Now, K is also an h-clique in G', and $\{u\}$ is a 1-clique in G' disjoint from K. Therefore $\langle G', h, 1 \rangle = f(\langle G, h \rangle) \in$ TWO-CLIQUE. Consider now the case $f(\langle G, h \rangle) \in$ TWO-CLIQUE. If $h = 1$, then clearly $\langle G, h \rangle \in$ CLIQUE. Let now $h > 1$. Then there is an h-clique K in $(V \cup \{u\}, E)$. Since u is not adjacent to any other vertex in V, u is not contained in the h-clique. Therefore K is also an h-clique in G. So $\langle G, h \rangle \in$ CLIQUE. Finally, f simply copies G and adds an extra node, therefore f is computable in linear time. $\square$

Exercise 1.13 Consider the following decision problems:

OMC (ODD-MAX-CLIQUE):
INSTANCE: $\langle G = (V, E) \rangle$, with G an undirected graph.

QUESTION: Is the maximum clique size odd?

EMC (EVEN-MAX-CLIQUE):
INSTANCE: $\langle G = (V, E) \rangle$, with G an undirected graph.

QUESTION: Is the maximum clique size even?

(a) Show that OMC $<_P$ EMC.

(b) Show that if EMC is NP-complete then OMC is NP-complete.

Answer:

(a) Let $G = (V, E)$ be an undirected graph, and let $G' = (V', E')$ be defined as follows:

$$\begin{aligned} V' &= V \cup \{\alpha\}, \ \ \alpha \notin V; \\ E' &= E \cup \{\{\alpha, v\} : v \in V\}. \end{aligned}$$

Let now $f(\langle G \rangle) = \langle G' \rangle$. Clearly, $f(\langle G \rangle)$ can be computed in time polynomial in $|V|$ and $|E|$. Let $M \subseteq V$ be a max-clique for G, and $M' \subseteq V'$ be a max-clique for G'. Then, the following two claims hold:

1. $\alpha \in M'$.
   If this were not the case, since $\{\alpha, u\} \in E'$ for each $u \in M'$, $M' \cup \{\alpha\}$ would be a clique of size strictly greater than M', a contradiction.

2. $|M'| = |M| + 1$.
   By Claim 1, $\alpha \in M'$ and $M' - \{\alpha\}$ is a clique for G. Hence

$$|M| \geq |M'| - 1.$$

   $M \cup \{\alpha\}$ is a clique for G'. Hence

$$|M'| \geq |M| + 1$$

From Claim 1 and Claim 2 we conclude that G has an odd max clique iff G' has an even max clique. This proves that f reduces OMC to EMC. Therefore OMC $<_P$ EMC.

(b) Suppose that EMC is NP-complete. Then, from Part **(a)** it follows that OMC $\in$ NP. Therefore, it is sufficient to show that EMC $<_P$ OMC, which requires an identical argument to the one used in Part **(a)**, since function f also reduces EMC to OMC. □

Exercise 1.14 Consider the following decision problem:

IS (INDEPENDENT SET):

INSTANCE:	$\langle G = (V, E), k \rangle$, with G an undirected graph, and $k > 0$.
QUESTION:	Is there a subset $S \subseteq V$, $|S| = k$, with $\{u, v\} \notin E$ for each $u, v \in S$?

(a) Show that IS is NP-Complete.

(b) Assume that you are given an $O(|V| + |E|)$ algorithm for IS. Show how to use the algorithm to determine the *maximum* size of an independent set in time $O((|V| + |E|) \log |V|)$.

Answer: In order to prove that IS $\in NP$, consider the following verification algorithm A.

$A(x, y)$
if $x \neq \langle G = (V, E), k \rangle, k > 0$
  then return 0
if $y \neq \langle U \rangle, U \subseteq V$
  then return 0
if $|U| = k$
 then if IS_INDEPENDENT(G, U)
   then return 1
return 0

Subroutine IS_INDEPENDENT(G, U) checks the adjacency list of G to make sure that U is an independent set. Clearly, L_A = IS. The length of an accepting certificate y is $O(|V|) = O(|x|)$. Finally, IS_INDEPENDENT(G, U) can clearly be implemented in polynomial time, therefore A is polynomial.

Next we show that CLIQUE $<_P$ IS, hence IS is NP-hard. Consider the following transformation:

$$f\left(\langle G = (V, E), k \rangle\right) = \langle G^c = (V, E^c), k \rangle,$$

where $E^c = \{\{u, v\} : u \neq v \in V \text{ and } \{u, v\} \notin E\}$. Then:

1. Since there is an edge (u, v) in G^c if and only if $(u, v) \notin E$, G^c can be determined by checking all the pairs of vertices in $O(|V|^2)$ time. Therefore f is computable in polynomial time.

2. If G contains a clique $U \subseteq V$ of size k, then no pair of vertices in U will be connected by an edge in G^c. Therefore U is an IS of size k for G^c.

3. If G^c has an IS U of size k, then any pair of distinct vertices in U will be connected by an edge in G, therefore U is a clique of size k for G.

(b) Let DECIDE_IS$(\langle G = (V, E), k \rangle)$ be our (unlikely) $O(|V| + |E|)$ algorithm that decides IS. Based on DECIDE_IS, we can write the following recursive algorithm:

MAX_SIZE$(\langle G = (V, E) \rangle, i, j)$
if $i = j$ **then return** i
$\mathit{middle} \leftarrow \lceil (i + j)/2 \rceil$
if DECIDE_IS$(\langle G = (V, E), \mathit{middle} \rangle)$
  then return MAX_SIZE$(\langle G = (V, E) \rangle, \mathit{middle}, j)$
  else return MAX_SIZE$(\langle G = (V, E) \rangle, i, \mathit{middle} - 1)$

When we call MAX_SIZE($\langle G = (V, E) \rangle, 1, |V|$), we basically perform a binary search on all possible cardinalities of an independent set. The correctness of the algorithm follows from the observation that there is an independent set of size h iff the size of the maximum independent set is $\geq h$. Therefore a binary search approach can be applied, yieding the desired running time of $O((|V| + |E|) \log |V|)$. □

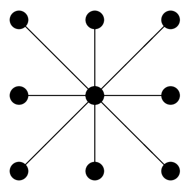
Exercise 1.15 A problem closely related to problem IS, defined in the previous exercise, is the following. Given an undirected graph $G = (V, E)$, a *maximal* independent set is an indepent set S such that, for each $v \in V - S$, $S \cup \{v\}$ is not independent. That is, S cannot be "upgraded" to a larger independent set.

(a) Give an example of a graph where there is a *maximal* independent set of size much smaller than the size of the *maximum* independent set.

(b) Show that the problem of determining a maximal independent set can be solved in polynomial time.

Answer:

(a) Consider the following "star" graph:



Clearly, the node at the center of the star makes a *maximal* independent set by itself, since all other nodes are connected to it. However, the *maximum* independent set contains eight nodes. Note that the above example can be generalized to yield a discrepancy of $\Theta(|V|)$ between the size of a a maximal and a maximum independent set, for any value of $|V|$.

(b) We build our maximal independent set S incrementally as follows. We start from the empty set and perform a linear scan the nodes. We add a new node v to S if $S \cup \{v\}$ is still independent. The algorithm follows.

```
GREEDY_MAXIMAL_INDEPENDENT_SET(G = (V, E))
n ← |V|
S ← ∅
for i ← 1 to n do
    indep ← true
    for each u ∈ Adj[v_i] do
        if u ∈ S
          then indep ← false
    if indep
      then S ← S ∪ {v_i}
return S
```

The set S returned by the above algorithm is an independent set by construction. Let us now prove that S is maximal. Assume, for the sake of contradiction, that S is not maximal. Then, there is a node $v_i \in V - S$ such that $S \cup \{v_i\}$ is an independent set. Note that v_i was not added to S, therefore, at the end of the i-th iteration of the outer loop, variable *indep* was **false**. This means that there was a node $u \in S$ such that $u \in Adj[v_i]$, which contradicts the hypothesis that $S \cup \{v_i\}$ is an independent set.

Note that the outer loop is executed $|V|$ times. During iteration i, we execute the inner loop $|Adj[v_i]|$ times, for a total of $\Theta(|E|)$ iterations altogether. In each iteration, the check $u \in S$ can be performed in $O(\log |S|)$ time (using –say– a binary search tree to store S). Since $|S| \leq |V|$, the running time of the above algorithm is then $O(|V| + |E| \log |V|)$. □

Exercise 1.16 Given undirected graphs $G_1 = (V(G_1), E(G_1))$ and $G_2 = (V(G_2, E(G_2))$, we say that G_1 is isomorphic to G_2 if there is a one-to-one function $\pi : V(G_1) \to V(G_2)$ such that $\{u, v\} \in E(G_1)$ iff $\{\pi(u), \pi(v)\} \in E(G_2)$. Consider the following decision problem:

SI (SUBGRAPH ISOMORPHISM):

INSTANCE: $\langle G = (V(G), E(G)), H = (V(H), E(H))\rangle$, with G and H undirected graphs

QUESTION: Does H contain a subgraph $H' = (V(H'), E(H'))$, with $V(H') \subseteq V(H)$ and $E(H') \subseteq E(H))$ that is isomorphic to G?

Show that SI is NP-complete.

Answer: SI is clearly in NP. Given a string $x = \langle G, H\rangle \in$ SI, a certificate y for SI is $\langle H' = (V(H'), E(H'), \pi\rangle$. Note that π can be represented as a sequence of $|V(G)|$ pairs $(u, \pi(u))$, with $u \in V(G)$, therefore the encoding of y is polynomial in the size of the instance. On input $\langle x, y\rangle$, the verifier first checks that the encodings for the instance and the certificate are well-formed, then checks that H' is indeed a subgraph of H with $|V(G)|$ nodes, and finally checks that for any edge (u, v) in $E(G)$, edge $(\pi(u), \pi(v))$ is in $E(H')$ and viceversa. These checks clearly take time polynomial in the size of $\langle x, y\rangle$. The code of the algorithm is omitted for the sake of brevity.

In order to show that SI is NP-Hard, we show that CLIQUE $<_P$ SI. Recall that an instance of CLIQUE is $\langle G, k\rangle$ and the question is whether G contains a complete subgraph of size k. Let C_k be the graph $(\{1, 2, \ldots, k\}, \{\{u, v\} : 1 \leq u \neq v \leq k\})$, that is, C_k is the complete graph built on vertices $V(C_k) = \{1, 2, \ldots, k\}$. Our reduction function is

$$f(\langle G, k\rangle) = \langle C_k, G\rangle.$$

Clearly, f is computable in polynomial time. To see that f reduces CLIQUE to SI, note that if G contains a complete subgraph with k nodes, then such subgraph is clearly isomorphic to C_k (all complete graphs with the same number of nodes are isomorphic). Viceversa, if G contains a subgraph isomorphic to C_k, then such subgraph is itself a clique of k nodes (a complete graph can only be isomorphic to another complete graph). This suffices to show that CLIQUE $<_P$ SI, and the claim follows. □

Exercise 1.17 Consider the following decision problem:

HS (HITTING SET):

INSTANCE:	$\langle n, m, C_1, C_2, \ldots, C_m, k\rangle$, with $C_i \subseteq \{1, 2, \ldots, n\}$ for $1 \leq i \leq m$, and $k \leq n$.
QUESTION:	Is there a subset $S' \subseteq \{1, 2, \ldots, n\}$ with $|S'| = k$ and such that $S' \cap C_i \neq \emptyset$, for $1 \leq i \leq m$?

Show that HS is NP-complete.

Answer: A candidate certificate for HS is a subset $S' \subseteq \{1, 2, \ldots, n\}$. The verification algorithm first checks whether its first input x is a well-formed encoding $x = \langle n, m, C_1, C_2, \ldots, C_m, k\rangle$ of an instance of HS; then checks that its second input encodes a subset of $\{1, 2, \ldots, n\}$ of cardinality k. If this is the case, the algorithm proceeds to check whether $S' \cap C_i \neq \emptyset$, for $1 \leq i \leq m$. Each such test can clearly be accomplished in polynomial time. Therefore HS $\in NP$.

In order to show that HS is NP-Hard, we exhibit a reduction from VERTEX_COVER (VC) to HS. Recall that an instance of VC is $\langle G = (V, E), k\rangle$ and the question is whether V contains a subset V' of size k such that each edge in E has at least one of its endpoints in V'.

Let $\pi : V \to \{1, 2, \ldots, |V|\}$ be an arbitrary one-to-one function from V to $\{1, 2, \ldots, |V|\}$. Our reduction function is the following:

$$f(\langle G = (V, E), k\rangle) = \langle |V|, |E|, C_1, C_2, \ldots, C_{|E|}, k\rangle,$$

where $C_i = \{\pi(u), \pi(v)\}$ iff the i-th edge in E is $\{u, v\}$.

Clearly, f is computable in polynomial time. To show that f reduces VC to HS, it is sufficient to observe that, by construction, G contains a vertex cover V' with k nodes if and only if $\pi(V') \subseteq \{1, 2, \ldots, |V|\}$ has nonempty intersection with all the C_i's. The proof follows since $|\pi(V')| = |V'| = k$. $\square$

Exercise 1.18 Given a language $L \in NP$, consider the following three cases.

(a) $L = \Sigma^*$

(b) $L \neq \emptyset, \Sigma^*$ is accepted by a DFSA.

(c) L contains an NP-complete subset.

Under the assumption $P \neq NP$, decide, for each of the above cases, whether 1) L is NP-complete or 2) L is not NP-complete or 3) L might be NP-complete or not. Redo the exercise under the assumption $P = NP$.

Exercise 1.19 Let $L_1, L_2 \in \{0, 1\}^\star$. Under the assumption that $P \neq NP$, prove or disprove the following propositions:

(a) $L_1 \in P \Rightarrow L_1^c \in NP$.

(b) $L_1 <_P L_2 \Leftrightarrow L_1^c <_P L_2^c$.

(c) $L_1 <_P L_{\text{SAT}} \Rightarrow L_1 \in NPC$.

(d) $L_1 <_P L_{\text{SAT}} \Rightarrow L_1 \in NP$.

(e) $L_1 <_P L_2$ and $L_2 <_P L_1 \Rightarrow L_1, L_2 \in P$.

(f) A reduction function f is a one-to-one correspondence.

(g) If we restricted the input set of CLIQUE to graphs $G = (V, E)$ of degree at most 7, then the resulting subproblem would be in P.

(h) If there is an algorithm for CLIQUE with running time $N^{O(\log N)}$, then every other problem in NP has an algorithm with a running time of the same form.

Exercise 1.20 Consider the following decision problem:

BF_SAT (BALANCED FORMULA SATISFIABILITY):
INSTANCE: $\langle \Phi(x_1, x_2, \ldots, x_{2n}) \rangle$, Φ is a boolean formula

QUESTION: Is there a satisfying assignment in which *exactly* n variables have value **false**?

Prove that BF_SAT is NP-Complete.

Exercise 1.21 Consider the following decision problem:

NCBF (NON CONSTANT BOOLEAN FORMULA):
INSTANCE: $\langle \Phi(x_1, x_2, \ldots, x_n) \rangle$, Φ is a boolean formula

QUESTION: Is $\Phi(x_1, x_2, \ldots, x_n)$ a *non constant* function? (i.e., $\Phi \not\equiv$ **false** and $\Phi \not\equiv$ **true**)

Show that NCBF is NP-Complete.

Exercise 1.22 Consider the following decision problem:

CoH (CLIQUE or HAMILTONIAN):
INSTANCE: $\langle G = (V, E), k \rangle$, with G an undirected graph and $k > 0$

QUESTION: Does G contain *either* a clique of size k *or* a hamiltonian circuit?

Show that CoH is NP-Complete.

Exercise 1.23 Consider the following decision problem:

RH (ROOT-HAMILTONIAN):
INSTANCE: $\langle G = (V, E) \rangle$, with G an undirected graph

QUESTION: Does G contain a simple cycle of length at least $\left\lceil \sqrt{|V|} \right\rceil$?

Show that RH is NP-Complete.

Exercise 1.24 Given an undirected graph G, recall that a *hamiltonian path* is a simple path that touches all nodes of G. Consider the following two problems:

HP (HAMILTONIAN PATH):
INSTANCE: $\langle G = (V, E) \rangle$, with G an undirected graph

QUESTION: Does G contain a hamiltonian path?

k-P (k-PATH):

INSTANCE: $\langle G, u, v, k \rangle$, with $G = (V, E)$ an undirected graph, $u \neq v \in V$ and $k > 0$

QUESTION: Does G contain a simple path containing at least k edges from u to v ?

(a) Show that HP is NP-Complete.

(b) Show that k-P is NP-Complete.

(c) Show that HP and k-P are both in P when the graph G is restricted to be acyclic.