1. Let Q,Q' be problems with $Q \leqslant Q'$ and $Q' \leqslant Q$. We say that Q and Q' are polynomial-time equivalent. Prove/Disprove: Any two NP-Complete problems are polynomial-time equivalent.

Since Q and Q' are NP-Complete, they are in NP. Moreover, since they are NP-hard, we have $Q \leq Q'$ and $Q' \leq Q$ by definition of NP-hard-ness. So Q and Q' are polynomial-time equivaalent.

2. DOUBLE-SAT: Decide whether a Boolean formula has at least two satisfying assignments. Prove that DOUBLE-SAT is NP-Complete.

Different parts of an NP-completeness proof

1. DOUBLE-SAT is in NP: For any φ in Accept(DOUBLE-SAT), two different truth assignments of the variables of φ, for which φ evaluates to true construct a succinct certificate. Such a certificate can be verified in time polynomial in the number of variables plus the size of φ.

2. A reduction from a known NP-complete problem: Here, we show SAT $\leqslant$ DOUBLE-SAT. Let φ be an instance of SAT. Take a variable y not present in φ, and define $\varphi' = \varphi \wedge (y \vee y')$. Take φ' as the converted instance for DOUBLE-SAT. This reduction should satisfy two properties.

(a) Doable in polynomial time [Clear in this example]

(b) Correctness: φ is satisfiable if and only if φ' is double satisfiable.

If φ is satisfiable, you can take y = T or y = F.

If φ is not satisfiable, φ' cannot be satisfied (let alone double satisfied) irrespective of the truth assignment of y.

3. A CNF formula is called not-all-equal satisfiable if for some truth assignment of the variables, each clause has at least one true literal and at least one false literal.

NAESAT: Decide whether a Boolean formula in CNF is not-all-equal satisfiable.

Prove that NAESAT is NP-Complete.

NAESAT is in NP: A truth assignment with the stated property is a succint certificate for an instance in Accept(NAESAT).

NP-hard-ness: Use the reduction CNFSAT $\leqslant$ NAESAT as follows. Let φ be an instance for CNFSAT. Convert it to an instance φ' of NAESAT as follows. First, choose a variable y not in φ. Let $l_1 \vee l_2 \vee \cdots \vee l_k$ be a clause in φ. Convert this to the clause $l_1 \vee l_2 \vee \cdots \vee l_k \vee y$ for φ'.

Clearly, this construction can be done in polynomial time.

Correctness: We need to prove that φ is satisfiable if and only if φ' is not-all-equal satisfiable.

[$\Longrightarrow$] Let $t_1, t_2, \cdots, t_n$ be a satisfying truth assignment for φ. Take the truth assignment $t_1, t_2, \cdots, t_n$, F for φ'.

[$\Longleftarrow$] Let $t_1, t_2, \cdots, t_n, \theta$ be a truth assignment that not-all-equal satisfies φ'. If θ = F, then $t_1, t_2, \cdots, t_n$ satisfies φ. If θ = T, then $t'_1, t'_2, \cdots, t'_n$ is a satisfying truth assignment for φ.

4. Let $\Phi(x_1, x_2, \ldots, x_n)$ be a Boolean formula in the conjunctive normal form (CNF). We say that Φ is all-but-one satisfiable if there is a truth assignment of the variables for which all except exactly one of the clauses of Φ evaluate to true. By AB1SAT, we denote the problem of deciding whether the given CNF formula Φ is all-but-one satisfiable.

An appropriate truth assignment of the variables is a succinct certificate for an instance in Accept(AB1SAT). The CNF formula can be evaluated in time linear in the size of Φ plus n. It is an easy matter to check how many clauses are satisfied for the given truth assignment.

We reduce CNFSAT to AB1SAT. Let $\Phi(x_1, x_2, \ldots, x_n)$ be an instance for CNFSAT. We introduce a new variable y, and generate the CNF formula $\hat{\Phi}(x_1, x_2, \ldots, x_n, y) = \Phi(x_1, x_2, \ldots, x_n) \wedge y \wedge y'$. We show that Φ is satisfiable if and only if $\hat{\Phi}$ is all-but-one satisfiable. If Φ is satisfiable by some truth assignment of $x_1, x_2, \ldots, x_n$, then the same truth assignment along with any truth assignment for y all-but-one satisfies $\hat{\Phi}$. On the other hand, if Φ is not satisfiable, then for each truth assignment of $x_1, x_2, \ldots, x_n$, at least one clause of Φ evaluates to false. Moreover, one of the clauses y and y' must be false too. So $\hat{\Phi}$ cannot be all-but-one satisfied.

This reduction shows that AB1SAT is NP-hard. By Part (a), it is NP-complete too.

5. **[Subgraph isomorphism problem]** Given two graphs G and H, decide whether there exists an injective function $f : \mathrm{V}(H) \to \mathrm{V}(G)$ such that

$$(u,v) \in \mathrm{E}(H) \text{ if and only if } (f(u), f(v)) \in \mathrm{E}(G).$$

Prove that SUBGRAPH-ISOMORPHISM is NP-complete.

Solution

First, I show that SUBGRAPH-ISOMORPHISM is in NP. For an input (G,H) of SUBGRAPH-ISOMORPHISM, guess a subset $V' \subseteq V(G)$ together with a bijective function $f : V' \to V(H)$. Then, check whether f preserves the adjacency relations of V' in $V(H)$.

In order to show the NP-Hardness, I reduce CLIQUE to SUBGRAPH-ISOMORPHISM. Let (G,r) be an input for CLIQUE. We construct the input (G,K_r) for SUBGRAPH-ISOMORPHISM, where K_r is the complete graph on r vertices. G has an r-clique if and only if G has a subgraph isomorphic to K_r. This reduction evidently runs in polynomial time.

6. **[Independent Set Problem]** Let $G = (V, E)$ be an undirected graph. A subset U of V is called an independent set if for all $u,v \in U$, we have $(u,v) \notin E$. The independent set problem takes G and a positive integer k as input, and decides whether G contains an independent set with k vertices. Prove that INDEPENDENT-SET is NP-complete.

Clearly, INDEPENDENT-SET is in NP (an independent set of size k is the certificate). For proving the NP-hardness of this problem, we reduce from CLIQUE. Let (G, k) be an instance of CLIQUE. Convert it to the instance (G', k) as follows. Let $G = (V, E)$. Take $G' = (V, E')$, where E' is the complement of E, that is, an edge $(u,v) \in E'$ if and only if $(u,v) \notin E$. Clearly, a clique is converted to an independent set by the complementing process, and conversely.

7. **[Vertex Cover Problem]** Let $G = (V, E)$ be an undirected graph. A subset U of V is called a vertex cover of G if for all $(u,v) \in E$, we have either $u \in U$ or $v \in U$ (or both). The vertex cover problem takes G and a positive integer k as input, and decides whether G contains a vertex cover with k vertices. Prove that VERTEX-COVER is NP-complete.

Clearly, VERTEX-COVER is in NP (a vertex cover of the desired size is the certificate). In order to prove the NP-hardness of this problem, we reduce from INDEPENDENT-SET. Let (G, k) be an instance for INDEPENDENT-SET. Convert this to the instance $(G, n - k)$ for VERTEX-COVER, where $n = |V|$. It is easy to see that U is an independent set in G if and only if $V - U$ is a vertex cover of G.

8. Prove that if P = NP, then every non-trivial problem in this class is NP-complete.

Solution

Let P be a non-trivial problem in P = NP. We want to show that P in NP-Complete, that is, every problem $Q \in$ NP reduces to P in polynomial time. Let I_1 be a fixed instance for P for which the answer is *Yes*, and I_2 a fixed instance for P for which the answer is *No*. Finally, let I be an input instance for Q. Since NP = P, there exists a polynomial-time algorithm to solve Q. Invoke this algorithm to solve Q on I. If the answer is *Yes* (respectively, *No*), the reduction algorithm generates the instance I_1 (respectively, I_2) for P. Since I_1 and I_2 are fixed instances, their sizes are constant, that is, do not depend on the size of I. Therefore the running time of the reduction algorithm is the same as that to solve Q on I. This shows that $Q \leqslant P$.

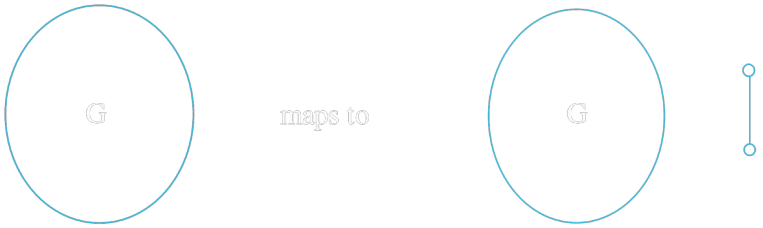
9. Let P and Q be two problems in NP such that the same polynomial-time reduction f can be used for proving both $P \leqslant Q$ and $Q \leqslant P$. Prove/Disprove: f must be a bijection.

False. Let G be an undirected graph. Consider the following problems on G.

EVC: Decide whether G has a minimal vertex cover of even size
OVC: Decide whether G has a minimal vertex cover of odd size

(The covers are minimal in the sense that removing any vertex from the cover fails to cover all the edges. Minimum covers are minimal, but not necessarily conversely. These problems are clearly in NP.)

Consider the following reduction that works for both EVC $\leqslant$ OVC and OVC $\leqslant$ EVC. This is clearly not a bijection (every application increases the number of vertices by two).



G maps to G'

10. Every instance of a problem in NP can be encoded in binary. Without loss of generality, we can therefore assume that the space of input instances of all problems in NP is $\{0,1\}^*$. Invalid encodings can be assumed to belong to the REJECT set.

The intersection $P \wedge Q$ of two problems P and Q in NP is the problem having

$$\text{Accept}(P \wedge Q) = \text{Accept}(P) \cap \text{Accept}(Q).$$

(a) Prove that the class NP is closed under intersection.

A cetificate for I in Accept($P \wedge Q$) is a certificate of I in Accept(P) concatenated with a certificate of I in Accept(Q).

(b) Prove that the class of NP-complete problems is not closed under intersection.

Let G be an undirected graph with n vertices. Consider the following problems.

LARGE_CLIQUE: Decide whether G has a clique of size $\geq n / 2 + 1$
LARGE_IS: Decide whether G has an independent set of size $\geq n / 2 + 1$

First, prove that these problems are NP-complete (use reductions from CLIQUE and IND_SET).

Then, note that the intersection of these problems is trivial, that is,
Accept($P \wedge Q$) = $\varnothing$ and Reject($P \wedge Q$) = $\{0, 1\}^*$.

11. Suppose that we want to factor a positive integer n (may be assumed to be composite). This is not a decision problem. Formulate a decision problem that can be solved in polynomial time if and only if the factoring can be solved in polynomial time.

Decision problem: Given n and k, decide whether n has a factor $\leqslant k$.

Clearly, if we can factor n in polynomial time, then we can solve the decision problem (just check whether the smallest prime factor of n is $\leqslant k$).

Conversely, suppose that the decision problem has a polynomial-time algorithm. Invoke this algorithm multiple times to stage a binary search in order to detect a non-trivial divisor d of n. The binary search may be so designed that it gives the smallest divisor d of n. But then, d must be a prime. Now, recursively factor n / d. This factoring algorithm runs in polynomial time, because n can have only $O(\log n)$ prime divisors.