34 NP-Completeness

34.1 Polynomial time

34.1-1

> Define the optimization problem $\text{LONGEST-PATH-LENGTH}$ as the relation that associates each instance of an undirected graph and two vertices with the number of edges in a longest simple path between the two vertices. Define the decision problem $\text{LONGEST-PATH} = \{\langle G, u, v, k\rangle : G = (V, E)$ is an undirected graph, $u, v \in V, k \geq 0$ is an integer, and there exists a simple path from u to v in G consisting of at least k edges $\}$. Show that the optimization problem $\text{LONGEST-PATH-LENGTH}$ can be solved in polynomial time if and only if $\text{LONGEST-PATH} \in P$.

Showing that $\text{LONGEST-PATH-LENGTH}$ being polynomial implies that LONGEST-PATH is polynomial is trivial, because we can just compute the length of the longest path and reject the instance of LONGEST-PATH if and only if k is larger than the number we computed as the length of the longest path.

Since we know that the number of edges in the longest path length is between 0 and $|E|$, we can perform a binary search for it's length. That is, we construct an instance of LONGEST-PATH with the given parameters along with $k = \frac{|E|}{2}$. If we hear yes, we know that the length of the longest path is somewhere above the halfway point. If we hear no, we know it is somewhere below. Since each time we are halving the possible range, we have that the procedure can require $O(\lg |E|)$ many steps. However, running a polynomial time subroutine $\lg n$ many times still gets us a polynomial time procedure, since we know that with this procedure we will never be feeding output of one call of LONGEST-PATH into the next.

34.1-2

> Give a formal definition for the problem of finding the longest simple cycle in an undirected graph. Give a related decision problem. Give the language corresponding to the decision problem.

The problem $\text{LONGST-SIMPLE-CYCLE}$ is the relation that associates each instance of a graph with the longest simple cycle contained in that graph. The decision problem is, given k, to determine whether or not the instance graph has a simple cycle of length at least k. If yes, output 1. Otherwise output 0. The language corresponding to the decision problem is the set of all $\langle G, k\rangle$ such that $G = (V, E)$ is an undirected graph, $k \geq 0$ is an integer, and there exists a simple cycle in G consisting of at least k edges.

34.1-3

> Give a formal encoding of directed graphs as binary strings using an adjacencymatrix representation. Do the same using an adjacency-list representation. Argue that the two representations are polynomially related.

(Omit!)

34.1-4

> Is the dynamic-programming algorithm for the 0-1 knapsack problem that is asked for in Exercise 16.2-2 a polynomial-time algorithm? Explain your answer.

This isn't a polynomial-time algorithm. Recall that the algorithm from Exercise 16.2-2 had running time $\Theta(nW)$ where W was the maximum weight supported by the knapsack. Consider an encoding of the problem. There is a polynomial encoding of each item by giving the binary representation of its index, worth, and weight, represented as some binary string of length $a = \Omega(n)$. We then encode W, in polynomial time. This will have length $\Theta(\lg W) = b$. The solution to this problem of length $a + b$ is found in time $\Theta(nW) = \Theta(a \cdot 2^b)$. Thus, the algorithm is actually exponential.

34.1-5

> Show that if an algorithm makes at most a constant number of calls to polynomial-time subroutines and performs an additional amount of work that also takes polynomial time, then it runs in polynomial time. Also show that a polynomial number of calls to polynomial-time subroutines may result in an exponential-time algorithm.

(Omit!)

34.1-6

> Show that the class P, viewed as a set of languages, is closed under union, intersection, concatenation, complement, and Kleene star. That is, if $L_1, L_2 \in P$, then $L_1 \cup L_2 \in P$, $L_1 \cap L_2 \in P$, $L_1 L_2 \in P$, $\overline{L_1} \in P$, and $L_1^* \in P$.

(Omit!)

34.2 Polynomial-time verification

34.2-1

> Consider the language $\text{GRAPH-ISOMORPHISM} = \{\langle G_1, G_2 \rangle : G_1 \text{ and } G_2 \text{ are isomorphic graphs}\}$. Prove that $\text{GRAPH-ISOMORPHISM} \in \text{NP}$ by describing a polynomial-time algorithm to verify the language.

(Omit!)

34.2-2

> Prove that if G is an undirected bipartite graph with an odd number of vertices, then G is nonhamiltonian.

(Omit!)

34.2-3

> Show that if $\text{HAM-CYCLE} \in \text{P}$, then the problem of listing the vertices of a hamiltonian cycle, in order, is polynomial-time solvable.

(Omit!)

34.2-4

> Prove that the class NP of languages is closed under union, intersection, concatenation, and Kleene star. Discuss the closure of NP under complement.

(Omit!)

34.2-5

> Show that any language in NP can be decided by an algorithm running in time $2^{O(n^k)}$ for some constant k.

(Omit!)

34.2-6

> A ***hamiltonian path*** in a graph is a simple path that visits every vertex exactly once. Show that the language $\text{HAM-PATH} = \{\langle G, u, v\rangle : \text{there is a hamiltonian path from } u \text{ to } v \text{ in graph } G\}$ belongs to NP.

(Omit!)

34.2-7

> Show that the hamiltonian-path problem from Exercise 34.2-6 can be solved in polynomial time on directed acyclic graphs. Give an efficient algorithm for the problem.

(Omit!)

34.2-8

> Let ϕ be a boolean formula constructed from the boolean input variables $x_1, x_2, \ldots, x_k$, negations ($\neg$), ANDs ($\vee$), ORs ($\wedge$), and parentheses. The formula ϕ is a ***tautology*** if it evaluates to 1 for every assignment of 1 and 0 to the input variables. Define TAUTOLOGY as the language of boolean formulas that are tautologies. Show that $\text{TAUTOLOGY} \in \text{co-NP}$.

(Omit!)

34.2-9

> Prove that $\mathrm{P} \subseteq \text{co-NP}$.

(Omit!)

34.2-10

> Prove that if $\mathrm{NP} \neq \text{co-NP}$, then $\mathrm{P} \neq \mathrm{NP}$.

(Omit!)

34.2-11

> Let G be a connected, undirected graph with at least 3 vertices, and let G^3 be the graph obtained by connecting all pairs of vertices that are connected by a path in G of length at most 3. Prove that G^3 is hamiltonian. (Hint: Construct a spanning tree for G, and use an inductive argument.)

(Omit!)

34.3 NP-completeness and reducibility

34.3-1

> Verify that the circuit in Figure 34.8(b) is unsatisfiable.

(Omit!)

34.3-2

> Show that the $\leq_{\mathrm{P}}$ relation is a transitive relation on languages. That is, show that if $L_1 \leq_{\mathrm{P}} L_2$ and $L_2 \leq_{\mathrm{P}} L_3$, then $L_1 \leq_{\mathrm{P}} L_3$.

(Omit!)

34.3-3

> Prove that $L \leq_{\mathrm{P}} \overline{L}$ if and only if $\overline{L} \leq_{\mathrm{P}} L$.

(Omit!)

34.3-4

> Show that we could have used a satisfying assignment as a certificate in an alternative proof of Lemma 34.5. Which certificate makes for an easier proof?

(Omit!)

34.3-5

> The proof of Lemma 34.6 assumes that the working storage for algorithm A occupies a contiguous region of polynomial size. Where in the proof do we exploit this assumption? Argue that this assumption does not involve any loss of generality.

(Omit!)

34.3-6

> A language L is ***complete*** for a language class C with respect to polynomial-time reductions if $L \in C$ and $L' \leq_P L$ for all $L' \in C$. Show that $\emptyset$ and $\{0,1\}^*$ are the only languages in P that are not complete for P with respect to polynomial-time reductions.

(Omit!)

34.3-7

> Show that, with respect to polynomial-time reductions (see Exercise 34.3-6), L is complete for NP if and only if $\overline{L}$ is complete for \text{co-NP}$.

(Omit!)

34.3-8

> The reduction algorithm F in the proof of Lemma 34.6 constructs the circuit $C = f(x)$ based on knowledge of x, A, and k. Professor Sartre observes that the string x is input to F, but only the existence of A, k, and the constant factor implicit in the $O(n^k)$ running time is known to F (since the language L belongs to NP), not their actual values. Thus, the professor concludes that F can't possibly construct the circuit C and that the language $CIRCUIT\text{-}SAT$ is not necessarily $NP\text{-}hard$. Explain the flaw in the professor's reasoning.

(Omit!)

34.4 NP-completeness proofs

34.4-1

> Consider the straightforward (nonpolynomial-time) reduction in the proof of Theorem 34.9. Describe a circuit of size n that, when converted to a formula by this method, yields a formula whose size is exponential in n.

(Omit!)

34.4-2

> Show the $3\text{-}CNF$ formula that results when we use the method of Theorem 34.10 on the formula (34.3).

(Omit!)

34.4-3

> Professor Jagger proposes to show that $\text{SAT} \leq_{\text{P}} \text{3-CNF-SAT}$ by using only the truth-table technique in the proof of Theorem 34.10, and not the other steps. That is, the professor proposes to take the boolean formula ϕ, form a truth table for its variables, derive from the truth table a formula in 3-DNF that is equivalent to $\neg\phi$, and then negate and apply DeMorgan's laws to produce a 3-CNF formula equivalent to ϕ. Show that this strategy does not yield a polynomial-time reduction.

(Omit!)

34.4-4

> Show that the problem of determining whether a boolean formula is a tautology is complete for co-NP. (Hint: See Exercise 34.3-7.)

(Omit!)

34.4-5

> Show that the problem of determining the satisfiability of boolean formulas in disjunctive normal form is polynomial-time solvable.

(Omit!)

34.4-6

> 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.

(Omit!)

34.4-7

> Let 2-CNF-SAT be the set of satisfiable boolean formulas in CNF with exactly 2 literals per clause. Show that $2\text{-CNF-SAT} \in \text{P}$. Make your algorithm as efficient as possible. (Hint: Observe that $\text{x} \vee \text{y}$ is equivalent to $\neg\text{x} \rightarrow \text{y}$. Reduce 2-CNF-SAT to an efficiently solvable problem on a directed graph.)

(Omit!)

34.5 NP-complete problems

34.5-1

> The ***subgraph-isomorphism problem*** takes two undirected graphs G_1 and G_2, and it asks whether G_1 is isomorphic to a subgraph of G_2. Show that the subgraphisomorphism problem is NP-complete.

(Omit!)

34.5-2

> Given an integer $m \times n$ matrix A and an integer m-vector b, the ***0-1 integerprogramming problem*** asks whether there exists an integer n-vector x with elements in the set $\{0, 1\}$ such that $Ax \leq b$. Prove that 0-1 integer programming is NP-complete. (Hint: Reduce from 3-CNF-SAT.)

(Omit!)

34.5-3

> The integer ***linear-programming problem*** is like the 0-1 integer-programming problem given in Exercise 34.5-2, except that the values of the vector x may be any integers rather than just 0 or 1. Assuming that the 0-1 integer-programming problem is NP-hard, show that the integer linear-programming problem is NP-complete.

(Omit!)

34.5-4

> Show how to solve the subset-sum problem in polynomial time if the target value t is expressed in unary.

(Omit!)

34.5-5

> The ***set-partition problem*** takes as input a set S of numbers. The question is whether the numbers can be partitioned into two sets A and $\bar{A} = S - A$ such that $\sum_{x \in A} x = \sum_{x \in \bar{A}} x$. Show that the set-partition problem is NP-complete.

(Omit!)

34.5-6

> Show that the hamiltonian-path problem is NP-complete.

(Omit!)

34.5-7

> The ***longest-simple-cycle problem*** is the problem of determining a simple cycle (no repeated vertices) of maximum length in a graph. Formulate a related decision problem, and show that the decision problem is NP-complete.

(Omit!)

34.5-8

> In the ***half 3-CNF satisfiability*** problem, we are given a 3-CNF formula ϕ with n variables and m clauses, where m is even. We wish to determine whether there exists a truth assignment to the variables of ϕ such that exactly half the clauses evaluate to 0 and exactly half the clauses evaluate to 1. Prove that the half 3-CNF satisfiability problem is NP-complete.

(Omit!)

Problem 34-1 Independent set

> An ***independent set*** of a graph $G = (V, E)$ is a subset $V' \subseteq V$ of vertices such that each edge in E is incident on at most one vertex in V'. The ***independent-set problem*** is to find a maximum-size independent set in G.
>
> **a.** Formulate a related decision problem for the independent-set problem, and prove that it is NP-complete. (Hint: Reduce from the clique problem.)
>
> **b.** Suppose that you are given a "black-box" subroutine to solve the decision problem you defined in part (a). Give an algorithm to find an independent set of maximum size. The running time of your algorithm should be polynomial in $|V|$ and $|E|$, counting queries to the black box as a single step.
>
> Although the independent-set decision problem is NP-complete, certain special cases are polynomial-time solvable.
>
> **c.** Give an efficient algorithm to solve the independent-set problem when each vertex in G has degree 2. Analyze the running time, and prove that your algorithm works correctly.
>
> **d.** Give an efficient algorithm to solve the independent-set problem when G is bipartite. Analyze the running time, and prove that your algorithm works correctly. (Hint: Use the results of Section 26.3.)

(Omit!)

Problem 34-2 Bonnie and Clyde

> Bonnie and Clyde have just robbed a bank. They have a bag of money and want to divide it up. For each of the following scenarios, either give a polynomial-time algorithm, or prove that the problem is NP-complete. The input in each case is a list of the n items in the bag, along with the value of each.
>
> **a.** The bag contains n coins, but only 2 different denominations: some coins are worth x dollars, and some are worth y dollars. Bonnie and Clyde wish to divide the money exactly evenly.
>
> **b.** The bag contains n coins, with an arbitrary number of different denominations, but each denomination is a nonnegative integer power of 2, i.e., the possible denominations are 1 dollar, 2 dollars, 4 dollars, etc. Bonnie and Clyde wish to divide the money exactly evenly.
>
> **c.** The bag contains n checks, which are, in an amazing coincidence, made out to "Bonnie or Clyde." They wish to divide the checks so that they each get the exact same amount of money.

> **d.** The bag contains n checks as in part (c), but this time Bonnie and Clyde are willing to accept a split in which the difference is no larger than 100 dollars.

(Omit!)

Problem 34-3 Graph coloring

> Mapmakers try to use as few colors as possible when coloring countries on a map, as long as no two countries that share a border have the same color. We can model this problem with an undirected graph $G = (V, E)$ in which each vertex represents a country and vertices whose respective countries share a border are adjacent. Then, a ***k-coloring*** is a function $c : V \to \{1, 2, \ldots, k\}$ such that $c(u) \neq c(v)$ for every edge $(u, v) \in E$. In other words, the numbers $1, 2, \ldots, k$ represent the k colors, and adjacent vertices must have different colors. The ***graph-coloring problem*** is to determine the minimum number of colors needed to color a given graph.
>
> **a.** Give an efficient algorithm to determine a 2-coloring of a graph, if one exists.
>
> **b.** Cast the graph-coloring problem as a decision problem. Show that your decision problem is solvable in polynomial time if and only if the graph-coloring problem is solvable in polynomial time.
>
> **c.** Let the language 3-COLOR be the set of graphs that can be 3-colored. Show that if 3-COLOR is NP-complete, then your decision problem from part (b) is NP-complete.
>
> To prove that 3-COLOR is NP-complete, we use a reduction from 3-CNF-SAT. Given a formula ϕ of m clauses on n variables $x_1, x_2, \ldots, x_n$, we construct a graph $G = (V, E)$ as follows. The set V consists of a vertex for each variable, a vertex for the negation of each variable, 5 vertices for each clause, and 3 special vertices: TRUE, FALSE, and RED. The edges of the graph are of two types: "literal" edges that are independent of the clauses and "clause" edges that depend on the clauses. The literal edges form a triangle on the special vertices and also form a triangle on x_i, $\neg x_i$, and RED for $i = 1, 2, \ldots, n$.
>
> **d.** Argue that in any 3-coloring c of a graph containing the literal edges, exactly one of a variable and its negation is colored $c(\text{TRUE})$ and the other is colored $c(\text{FALSE})$. Argue that for any truth assignment for ϕ, there exists a 3-coloring of the graph containing just the literal edges.
>
> The widget shown in Figure 34.20 helps to enforce the condition corresponding to a clause $(x \vee y \vee z)$. Each clause requires a unique copy of the 5 vertices that are heavily shaded in the figure; they connect as shown to the literals of the clause and the special vertex TRUE.
>
> **e.** Argue that if each of x, y, and z is colored $c(\text{TRUE})$ or $c(\text{FALSE})$, then the widget is 3-colorable if and only if at least one of x, y, or z is colored $c(\text{TRUE})$.
>
> **f.** Complete the proof that 3-COLOR is NP-complete.

(Omit!)

Problem 34-4 Scheduling with profits and deadlines

> Suppose that we have one machine and a set of n tasks $a_1, a_2, \ldots, a_n$, each of which requires time on the machine. Each task a_j requires t_j time units on the machine (its processing time), yields a profit of p_j, and has a deadline d_j. The machine can process only one task at a time, and task a_j must run without interruption for t_j consecutive time units. If we complete task a_j by its deadline d_j, we receive a profit p_j, but if we complete it after its deadline, we receive no profit. As an optimization problem, we are given the processing times, profits, and deadlines for a set of n tasks, and we wish to find a schedule that completes all the tasks and returns the greatest amount of profit. The processing times, profits, and deadlines are all nonnegative numbers.
>
> **a.** State this problem as a decision problem.
>
> **b.** Show that the decision problem is NP-complete.
>
> **c.** Give a polynomial-time algorithm for the decision problem, assuming that all processing times are integers from 1 to n. (Hint: Use dynamic programming.)
>
> **d.** Give a polynomial-time algorithm for the optimization problem, assuming that all processing times are integers from 1 to n.

(Omit!)