Exercises

8.1. *Optimization versus search.* Recall the traveling salesman problem:

> TSP
> *Input:* A matrix of distances; a budget b
> *Output:* A tour which passes through all the cities and has length $\leq b$, if such a tour exists.

The optimization version of this problem asks directly for the shortest tour.

> TSP-OPT
> *Input:* A matrix of distances
> *Output:* The shortest tour which passes through all the cities.

Show that if TSP can be solved in polynomial time, then so can TSP-OPT.

8.2. *Search versus decision.* Suppose you have a procedure which runs in polynomial time and tells you whether or not a graph has a Rudrata path. Show that you can use it to develop a polynomial-time algorithm for RUDRATA PATH (which returns the actual path, if it exists).

8.3. STINGY SAT is the following problem: given a set of clauses (each a disjunction of literals) and an integer k, find a satisfying assignment in which at most k variables are `true`, if such an assignment exists. Prove that STINGY SAT is **NP**-complete.

8.4. Consider the CLIQUE problem restricted to graphs in which every vertex has degree at most 3. Call this problem CLIQUE-3.

(a) Prove that CLIQUE-3 is in **NP**.

(b) What is wrong with the following proof of **NP**-completeness for CLIQUE-3?
We know that the CLIQUE problem in general graphs is **NP**-complete, so it is enough to present a reduction from CLIQUE-3 to CLIQUE. Given a graph G with vertices of degree ≤ 3, and a parameter g, the reduction leaves the graph and the parameter unchanged: clearly the output of the reduction is a possible input for the CLIQUE problem. Furthermore, the answer to both problems is identical. This proves the correctness of the reduction and, therefore, the **NP**-completeness of CLIQUE-3.

(c) It is true that the VERTEX COVER problem remains **NP**-complete even when restricted to graphs in which every vertex has degree at most 3. Call this problem VC-3. What is wrong with the following proof of **NP**-completeness for CLIQUE-3?
We present a reduction from VC-3 to CLIQUE-3. Given a graph $G = (V, E)$ with node degrees bounded by 3, and a parameter b, we create an instance of CLIQUE-3 by leaving the graph unchanged and switching the parameter to $|V| - b$. Now, a subset $C \subseteq V$ is a vertex cover in G if and only if the complementary set $V - C$ is a clique in G. Therefore G has a vertex cover of size $\leq b$ if and only if it has a clique of size $\geq |V| - b$. This proves the correctness of the reduction and, consequently, the **NP**-completeness of CLIQUE-3.

(d) Describe an $O(|V|^4)$ algorithm for CLIQUE-3.

8.5. Give a simple reduction from 3D MATCHING to SAT, and another from RUDRATA CYCLE to SAT. (*Hint:* In the latter case you may use variables x_{ij} whose intuitive meaning is "vertex i is the jth vertex of the Hamilton cycle"; you then need to write clauses that express the constraints of the problem.)

8.6. On page 266 we saw that 3SAT remains **NP**-complete even when restricted to formulas in which each literal appears at most twice.

(a) Show that if each literal appears at most *once*, then the problem is solvable in polynomial time.

(b) Show that INDEPENDENT SET remains **NP**-complete even in the special case when all the nodes in the graph have degree at most 4.

8.7. Consider a special case of 3SAT in which all clauses have exactly three literals, and each variable appears at most three times. Show that this problem can be solved in polynomial time. (*Hint:* create a bipartite graph with clauses on the left, variables on the right, and edges whenever a variable appears in a clause. Use Exercise 7.30 to show that this graph has a matching.)

8.8. In the EXACT 4SAT problem, the input is a set of clauses, each of which is a disjunction of exactly four literals, and such that each variable occurs at most once in each clause. The goal is to find a satisfying assignment, if one exists. Prove that EXACT 4SAT is NP-complete.

8.9. In the HITTING SET problem, we are given a family of sets $\{S_1, S_2, \ldots, S_n\}$ and a budget b, and we wish to find a set H of size $\leq b$ which intersects every S_i, if such an H exists. In other words, we want $H \cap S_i \neq \emptyset$ for all i.

Show that HITTING SET is **NP**-complete.

8.10. *Proving* **NP***-completeness by generalization.* For each of the problems below, prove that it is **NP**-complete by showing that it is a *generalization* of some **NP**-complete problem we have seen in this chapter.

(a) SUBGRAPH ISOMORPHISM: Given as input two undirected graphs G and H, determine whether G is a subgraph of H (that is, whether by deleting certain vertices and edges of H we obtain a graph that is, up to renaming of vertices, identical to G), and if so, return the corresponding mapping of $V(G)$ into $V(H)$.

(b) LONGEST PATH: Given a graph G and an integer g, find in G a simple path of length g.

(c) MAX SAT: Given a CNF formula and an integer g, find a truth assignment that satisfies at least g clauses.

(d) DENSE SUBGRAPH: Given a graph and two integers a and b, find a set of a vertices of G such that there are at least b edges between them.

(e) SPARSE SUBGRAPH: Given a graph and two integers a and b, find a set of a vertices of G such that there are at most b edges between them.

(f) SET COVER. (This problem generalizes *two* known **NP**-complete problems.)

(g) RELIABLE NETWORK: We are given two $n \times n$ matrices, a distance matrix d_{ij} and a *connectivity requirement* matrix r_{ij}, as well as a budget b; we must find a graph $G = (\{1, 2, \ldots, n\}, E)$ such that (1) the total cost of all edges is b or less and (2) between any two distinct vertices i and j there are r_{ij} vertex-disjoint paths. (*Hint:* Suppose that all d_{ij}'s are 1 or 2, $b = n$, and all r_{ij}'s are 2. Which well known **NP**-complete problem is this?)

8.11. There are many variants of Rudrata's problem, depending on whether the graph is undirected or directed, and whether a cycle or path is sought. Reduce the DIRECTED RUDRATA PATH problem to each of the following.

(a) The (undirected) RUDRATA PATH problem.

(b) The undirected RUDRATA (s,t)-PATH problem, which is just like RUDRATA PATH except that the endpoints of the path are specified in the input.

8.12. The k-SPANNING TREE problem is the following.

Input: An undirected graph $G = (V, E)$
Output: A spanning tree of G in which each node has degree $\leq k$, if such a tree exists.

Show that for any $k \geq 2$:

(a) k-SPANNING TREE is a search problem.

(b) k-SPANNING TREE is **NP**-complete. (*Hint:* Start with $k = 2$ and consider the relation between this problem and RUDRATA PATH.)

8.13. Determine which of the following problems are **NP**-complete and which are solvable in polynomial time. In each problem you are given an undirected graph $G = (V, E)$, along with:

(a) A set of nodes $L \subseteq V$, and you must find a spanning tree such that its set of leaves includes the set L.

(b) A set of nodes $L \subseteq V$, and you must find a spanning tree such that its set of leaves is precisely the set L.

(c) A set of nodes $L \subseteq V$, and you must find a spanning tree such that its set of leaves is included in the set L.

(d) An integer k, and you must find a spanning tree with k or fewer leaves.

(e) An integer k, and you must find a spanning tree with k or more leaves.

(f) An integer k, and you must find a spanning tree with exactly k leaves.

(*Hint:* All the **NP**-completeness proofs are by generalization, except for one.)

8.14. Prove that the following problem is **NP**-complete: given an undirected graph $G = (V, E)$ and an integer k, return a clique of size k *as well as* an independent set of size k, provided both exist.

8.15. Show that the following problem is **NP**-complete.

MAXIMUM COMMON SUBGRAPH
Input: Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$; a budget b.
Output: Two set of nodes $V_1' \subseteq V_1$ and $V_2' \subseteq V_2$ whose deletion leaves at least b nodes in each graph, and makes the two graphs identical.

8.16. We are feeling experimental and want to create a new dish. There are various ingredients we can choose from and we'd like to use as many of them as possible, but some ingredients don't go well with others. If there are n possible ingredients (numbered 1 to n), we write down an $n \times n$ matrix giving the *discord* between any pair of ingredients. This *discord* is a real number between 0.0 and 1.0, where 0.0 means "they go together perfectly" and 1.0 means "they really don't go together." Here's an example matrix when there are five possible ingredients.

	1	2	3	4	5
1	0.0	0.4	0.2	0.9	1.0
2	0.4	0.0	0.1	1.0	0.2
3	0.2	0.1	0.0	0.8	0.5
4	0.9	1.0	0.8	0.0	0.2
5	1.0	0.2	0.5	0.2	0.0

In this case, ingredients 2 and 3 go together pretty well whereas 1 and 5 clash badly. Notice that this matrix is necessarily symmetric; and that the diagonal entries are always 0.0. Any set of ingredients incurs a *penalty* which is *the sum of all discord values between pairs of ingredients*. For instance, the set of ingredients $\{1, 3, 5\}$ incurs a penalty of $0.2 + 1.0 + 0.5 = 1.7$. We want this penalty to be small.

> EXPERIMENTAL CUISINE
> *Input:* n, the number of ingredients to choose from; D, the $n \times n$ "discord" matrix; some number $p \geq 0$
> *Output:* The maximum number of ingredients we can choose with penalty $\leq p$.

Show that if EXPERIMENTAL CUISINE is solvable in polynomial time, then so is 3SAT.

8.17. Show that for any problem Π in **NP**, there is an algorithm which solves Π in time $O(2^{p(n)})$, where n is the size of the input instance and $p(n)$ is a polynomial (which may depend on Π).

8.18. Show that if $\mathbf{P} = \mathbf{NP}$ then the RSA cryptosystem (Section 1.4.2) can be broken in polynomial time.

8.19. A *kite* is a graph on an even number of vertices, say $2n$, in which n of the vertices form a clique and the remaining n vertices are connected in a "tail" that consists of a path joined to one of the vertices of the clique. Given a graph and a goal g, the KITE problem asks for a subgraph which is a kite and which contains $2g$ nodes. Prove that KITE is **NP**-complete.

8.20. In an undirected graph $G = (V, E)$, we say $D \subseteq V$ is a *dominating set* if every $v \in V$ is either in D or adjacent to at least one member of D. In the DOMINATING SET problem, the input is a graph and a budget b, and the aim is to find a dominating set in the graph of size at most b, if one exists. Prove that this problem is **NP**-complete.

8.21. *Sequencing by hybridization.* One experimental procedure for identifying a new DNA sequence repeatedly probes it to determine which k-mers (substrings of length k) it contains. Based on these, the full sequence must then be reconstructed.

Let's now formulate this as a combinatorial problem. For any string x (the DNA sequence), let $\Gamma(x)$ denote the multiset of all of its k-mers. In particular, $\Gamma(x)$ contains exactly $|x| - k + 1$ elements.

The reconstruction problem is now easy to state: given a multiset of k-length strings, find a string x such that $\Gamma(x)$ is exactly this multiset.

(a) Show that the reconstruction problem reduces to RUDRATA PATH. (*Hint:* Construct a directed graph with one node for each k-mer, and with an edge from a to b if the last $k - 1$ characters of a match the first $k - 1$ characters of b.)

(b) But in fact, there is much better news. Show that the same problem also reduces to EULER PATH. (*Hint:* This time, use one directed edge for each k-mer.)

8.22. In task scheduling, it is common to use a graph representation with a node for each task and a directed edge from task i to task j if i is a precondition for j. This directed graph depicts the precedence constraints in the scheduling problem. Clearly, a schedule is possible if and only if the graph is acyclic; if it isn't, we'd like to identify the smallest number of constraints that must be dropped so as to make it acyclic.

Given a directed graph $G = (V, E)$, a subset $E' \subseteq E$ is called a *feedback arc set* if the removal of edges E' renders G acyclic.

FEEDBACK ARC SET (FAS): Given a directed graph $G = (V, E)$ and a budget b, find a feedback arc set of $\leq b$ edges, if one exists.

(a) Show that FAS is in **NP**.

FAS can be shown to be **NP**-complete by a reduction from VERTEX COVER. Given an instance (G, b) of VERTEX COVER, where G is an undirected graph and we want a vertex cover of size $\leq b$, we construct a instance (G', b) of FAS as follows. If $G = (V, E)$ has n vertices $v_1, \ldots, v_n$, then make $G' = (V', E')$ a directed graph with $2n$ vertices $w_1, w'_1, \ldots, w_n, w'_n$, and $n + 2|E|$ (directed) edges:

- (w_i, w'_i) for all $i = 1, 2, \ldots, n$.
- (w'_i, w_j) and (w'_j, w_i) for every $(v_i, v_j) \in E$.

(b) Show that if G contains a vertex cover of size b, then G' contains a feedback arc set of size b.

(c) Show that if G' contains a feedback arc set of size b, then G contains a vertex cover of size (at most) b. (*Hint:* given a feedback arc set of size b in G', you may need to first modify it slightly to obtain another one which is of a more convenient form, but is of the same size or smaller. Then, argue that G must contain a vertex cover of the same size as the modified feedback arc set.)

8.23. In the NODE-DISJOINT PATHS problem, the input is an undirected graph in which some vertices have been specially marked: a certain number of "sources" $s_1, s_2, \ldots s_k$ and an equal number of "destinations" $t_1, t_2, \ldots t_k$. The goal is to find k node-disjoint paths (that is, paths which have no nodes in common) where the ith path goes from s_i to t_i. Show that this problem is **NP**-complete.

Here is a sequence of progressively stronger hints.

(a) Reduce from 3SAT.

(b) For a 3SAT formula with m clauses and n variables, use $k = m+n$ sources and destinations. Introduce one source/destination pair (s_x, t_x) for each variable x, and one source/destination pair (s_c, t_c) for each clause c.

(c) For each 3SAT clause, introduce 6 new intermediate vertices, one for each literal occurring in that clause and one for its complement.

(d) Notice that if the path from s_c to t_c goes through some intermediate vertex representing, say, an occurrence of variable x, then no other path can go through that vertex. What vertex would you like the other path to be forced to go through instead?