(b) State in English the complement of the problem HAM-PATH.

(c) Prove or disprove: DNF-SAT is the complement of CNF-SAT (See Exercise 6.7).

(d) A Boolean formula ϕ is called a *tautology* if it evaluates to *true* for all possible truth assignments of its variables. Prove that deciding whether a Boolean formula is a tautology is in coNP.

(e) A Boolean formula ϕ is called a *contradiction* if it evaluates to *false* for all possible truth assignments of its variables. Prove that deciding whether a Boolean formula is a contradiction is in coNP. What is its complement problem called?

6.33 Prove that $\mathrm{P} \subseteq \mathrm{NP} \cap \mathrm{coNP}$. (**Remark:** It is not known whether the containment is proper.)

6.34 Let SMALL-FACTOR denote the problem of deciding whether a given positive integer n has a factor no larger than a given positive integer k. Prove that SMALL-FACTOR is in $\mathrm{NP} \cap \mathrm{coNP}$.

6.35 Prove that if SMALL-FACTOR can be solved in polynomial time, integers can be factored in polynomial time too, and conversely.

6.36 Prove that if $\mathrm{NP} \neq \mathrm{coNP}$, then $\mathrm{P} \neq \mathrm{NP}$. (**Remark:** It is widely believed that $\mathrm{NP} \neq \mathrm{coNP}$.)

6.37 Prove that if an NP-Complete problem is in coNP, then $\mathrm{NP} = \mathrm{coNP}$.

6.38 Define coNP-Hard and coNP-Complete problems as those to which there exist polynomial-time reductions from *all* problems in coNP. Prove that a problem is coNP-Complete if and only if its complement is NP-Complete. State a few coNP-Complete problems.

6.39 Prove that the TAUTOLOGY problem (see Exercise 6.32(d)) is coNP-Complete.

6.40 Prove that the CONTRADICTION problem (see Exercise 6.32(e)) is coNP-Complete.

6.41 Prove or disprove: The halting problem is coNP-hard.

6.42 Let FSAT denote the problem of finding a satisfying assignment of a Boolean formula ϕ or return failure if ϕ is unsatisfiable. Prove that FSAT can be solved in polynomial-time if and only if $\mathrm{SAT} \in \mathrm{P}$.

Approximation algorithms

6.43 Consider the optimization version of the set covering problem of Exercise 6.13. That is, given a finite set S and a collection of k subsets $S_1, S_2, \ldots, S_k$ of S, we intend to find out a cover of S (from the given collection) of size as small as possible. Let us denote this optimization problem by MIN-SET-COVER.

Let $S = \{x_1, x_2, \ldots, x_n\}$ be of size n, and let f_i be the count of the subsets S_j containing the element x_i. Finally, let $f = \max(f_1, f_2, \ldots, f_n)$. (The counts f_i are the frequencies of the elements, and f is the maximum frequency.)

Design a polynomial-time f-approximation algorithm for MIN-SET-COVER. Establish that your algorithm achieves an approximation ratio of f. Determine whether this approximation ratio is tight.

6.44 Prof. Myopia proposes the following approximation algorithm for solving the MAX-CUT problem (Exercise 6.27).

1. Start with an arbitrary partition S, T of V.

2. Repeat the following two steps until no further vertex movement is possible:

   (a) For each vertex $v \in S$, check whether the cut $(S - v, T + v)$ has more cross edges than (S, T); and if so, delete v from S and include v in T.

   (b) For each vertex $v \in T$, check whether the cut $(S + v, T - v)$ has more cross edges than (S, T); and if so, delete v from T and include v in S.

3. Return S, T.

(a) Prove that Prof. Myopia's algorithm runs in polynomial time (in the input size).

(b) Prove or disprove: Prof. Myopia's algorithm outputs the optimal solution for bipartite graphs.

(c) Prove that the approximation ratio of Prof. Myopia's algorithm is 1/2.

(d) Demonstrate that this approximation ratio is tight (suggest an *infinite* family of graphs).

6.45 [*Weighted Max-Cut Problem*] Let $G = (V, E)$ be an undirected graph with each edge $e \in E$ having a (positive) cost c_e. The cost of a cut (S, T) (where S and T are disjoint subsets of V, and $S \cup T = V$) is $c(S, T) = \sum_{\substack{u \in S \\ v \in T}} c_{(u,v)}$. The task is to produce a cut (S, T) of G that maximizes $c(S, T)$. Adapt Prof. Myopia's algorithm (Exercise 6.44) to a $\frac{1}{2}$-approximation algorithm for the weighted max-cut problem.

6.46 Let $G = (V, E)$ be a connected undirected graph. You make a DFS traversal of G starting from any arbitrary vertex. Let T be the DFS tree produced by the traversal. Take C to be the set of all internal (that is, non-leaf) nodes in T.

(a) Prove that C is a vertex cover for G.

(b) Prove that the determination of C using this method is a 2-approximation algorithm for the MIN-VERTEX-COVER problem.

6.47 Adapt the approximation algorithm of Exercise 6.46 to a general undirected graph (not necessarily connected).

6.48 Consider the following algorithm for EUCLIDEAN TSP, that iteratively builds a closed tour C for the traveling salesperson. First, we choose two cities u, v such that $d(u, v)$ is smallest among all pairs of (different) cities. We start with $C = (u, v)$ (that is, go from u to v, and then back to u). Next, we run a loop, each iteration of which adds a new city to the currently constructed tour C. This city is chosen as follows. Find $i \in C$ and $j \notin C$ such that $d(i, j)$ is minimized. Let k be the city following i in the current tour C. Replace i, k by i, j, k in C. Repeat until C contains all the cities. Prove that this is a 2-approximation algorithm.

6.49 Consider the knapsack problem introduced in Section 25.1.4. First, sort the objects in decreasing order of p_i / w_i. Call this sorted list $O_1, O_2, \ldots, O_n$. Let k be the index such that $\sum_{i=1}^{k} w_i \leqslant C$ and $\sum_{i=1}^{k+1} w_i > C$. We have seen that the greedy algorithm that outputs $\{1, 2, \ldots, k\}$ can be arbitrarily bad. Now, let j be the index such that p_j is maximum. Augment the greedy algorithm to output $\{1, 2, \ldots, k\}$ or $\{j\}$, whichever gives better profit. Prove that this is a $1/2$-approximation algorithm for the knapsack problem.

6.50 Let us use the notations of Exercise 6.49. Modify the augmented greedy algorithm to output $\{1, 2, \ldots, k\}$ or $\{k+1\}$, whichever gives better profit. Prove that this is again a $1/2$-approximation algorithm for the knapsack problem.

6.51 Consider the balanced set partition problem introduced in Exercise 6.18.

(a) Prove that this problem cannot have any polynomial-time approximation algorithm (unless $\mathrm{P} = \mathrm{NP}$).

(b) We reformulate the problem as follows. Let A be the set with the larger sum. We want to minimize $\sum_{a\in A} a$ subject to the constraint that $\sum_{a\in A} a \geqslant \frac{1}{2}\sum_{x\in S} x$. Argue that there exists a trivial 2-approximation algorithm for this optimization problem.

6.52 Let $G = (V,E)$ be an undirected graph with each vertex v having a positive weight $w(v)$. The *weighted vertex cover problem* deals with the computation of a vertex cover C of G such that $\sum_{v\in C} w(v)$ is as small as possible.

(a) Prove that the 2-approximation algorithm of Section 25.1.1 fails for the weighted vertex cover problem, that is, given any positive constant Δ, there exists a graph for which the approximation factor is $> \Delta$.

(b) Prove that the 2-approximation algorithm of Exercise 6.47 fails too for the weighted vertex cover problem.

6.53 Assume that P $\neq$ NP. Prove that there cannot exist any ρ-approximation algorithm for the BIN-PACKING problem (Exercise 6.22) for $\rho < 3/2$.

6.54 The *next-fit* strategy for BIN-PACKING keeps on adding objects to the most recently opened bin so long as possible. When further placements are not possible, the last bin is closed, a new bin is opened, and the next object is put in the newly opened bin.

(a) Prove that the next-fit strategy is a 2-approximation algorithm.

(b) Prove that the approximation ratio 2 is tight, that is, given any $\varepsilon > 0$, there exists an input instance for which the algorithm achieves an approximation ratio $> 2-\varepsilon$.

6.55 The *first-fit* strategy for BIN-PACKING keeps all bins open. The objects are inserted in the sequence they appear in S. Each item is attempted to be inserted in the earliest open bin which can accommodate the object. If no opened bin can accommodate the object, a new bin is opened and the object is put in this newly opened bin. Prove that the first-fit strategy is a 2-approximation algorithm.

[H] **6.56** The *first-fit decreasing* strategy for BIN-PACKING first sorts the objects in the decreasing order of their weights, and inserts them in that sorted sequence following the first-fit strategy. Prove that this is a $3/2$-approximation algorithm.

6.57 The *linear bin-packing problem* deals with inserting the objects in the order they appear in the input. That is, for $i < j$, the j-th object cannot be packed in a bin opened earlier than the bin containing the i-th object. Argue that the next-fit strategy solves the linear bin-packing problem exactly.

[H] **6.58** Let $G = (V,E)$ be the complete undirected graph such that each pair (u,v) of vertices is associated with a distance $d(u,v) \geqslant 0$. Assume that $d(u,u) = 0$ for all u, $d(u,v) = d(v,u)$ for all u,v, and the distances satisfy the triangle inequality: $d(u,w) \leqslant d(u,v) + d(v,w)$ for all u,v,w. Let $S \subseteq V$ with $|S| = k$. For each $u \in V$, define $d(u,S) = \min_{v\in S} d(u,v)$. The *k-center problem* deals with the determination of a set S of size k such that $\max_{u\in V} d(u,S)$ is minimized. This problem is used for clustering the points in V with S playing the role of the set of cluster centers. Propose a 2-approximation algorithm for the k-center problem.

6.59 [*Makespan scheduling*] There are m identical machines, and n jobs with running times $t_1, t_2, \ldots, t_n$ assigned to the machines. The machines srart operation at time $t = 0$, and carry out the jobs assigned to them in a non-preemptive fashion without any waiting time between two consecutive jobs. The time when all the jobs finish is called the *makespan*. The task is to distribute the jobs to the machines in such a way that the makespan is minimized.

First sort $t_1, t_2, \ldots, t_n$ in non-increasing order, and assume that $t_1 \geqslant t_2 \geqslant \cdots \geqslant t_n \geqslant 0$. Now, for $i = 1, 2, \ldots, n$, assign the i-th job to the machine which finishes at the earliest (based upon the assignment of the first $i-1$ jobs). This is called *greedy makespan scheduling*. Deduce that this is a 2-approximation algorithm.

6.60 Suppose that $t_1, t_2, \ldots, t_n$ are not sorted as in the greedy makespan scheduling algorithm of Exercise 6.59. The rest of the algorithm remains the same. Prove that this is still a 2-approximation algorithm.

6.61 An algorithm is called *pseudo-polynomial-time*, if its running time is a polynomial in the size of the *unary* representation of the input. We call an NP-Complete problem *weakly NP-Complete* if it admits a pseudo-polynomial-time algorithm. Prove that the subset-sum problem (Exercise 6.14) is weakly NP-Complete.

6.62 Prove that the partition problem (Exercise 6.17) is weakly NP-Complete.

6.63 Prove that the balanced set partitioning problem (to be more precise, its decision version, see Exercise 6.18) is weakly NP-Complete.

6.64 Prove that the SUBSET-DIFF problem (Exercise 6.19) is weakly NP-Complete.

6.65 Prove that the 2-SET-SUM problem (Exercise 6.20) is weakly NP-Complete.

6.66 Prove that the 2-SET-PARTITION problem (Exercise 6.21) is weakly NP-Complete.

6.67 Prove that the ferry-loading problem (Exercise 6.24) is weakly NP-Complete.

[H^3] **6.68** An *asymptotic polynomial time approximation scheme* (*APTAS*) is a ρ-approximation algorithm such that $\rho \to 1+\varepsilon$ as OPT $\to \infty$ for some constant $\varepsilon > 0$. The best approximation ratio that we can achieve for BIN-PACKING is $3/2$ (Exercises 6.53 and 6.56). However, BIN-PACKING has an APTAS. More precisely, design an algorithm that given a constant $\varepsilon \in (0, 1/2]$ produces an output $m \leqslant (1+\varepsilon)\mathrm{OPT}+1$.

Randomized algorithms

6.69 Recall that a Las Vegas algorithm always outputs correct answers. Let us investigate a similar class of algorithms which may sometimes report *failure*. But whenever the algorithms succeed, the answer output is correct. Let us call such an algorithm a Las Vegas$'$ algorithm. Let A' be a Las Vegas$'$ algorithm for some problem. Using this algorithm, design a Las Vegas algorithm A that never outputs *failure*. Assume that A' outputs *failure* with probability $\leqslant 1/2$. Express the expected running time of A in terms of the expected running time of A'.

6.70 Suppose that a one-sided error Monte Carlo algorithm A has error probability $\leqslant 1/2$. Let $\delta > 0$ be a small real number. How many times you should run A in order to reduce the error probability to $< \delta$?

6.71 Consider the problem of finding the i-th smallest element in an array A of n integers. Ms. Lucky proposes the following algorithm to solve this problem. She chooses a (uniformly) random element x of A. She then uses the partitioning algorithm of Quick Sort on A with respect to the pivot x. Suppose that x is placed in the k-th position after the partitioning (counting starts from 1). If $k = i$, the algorithm returns x. If $k > i$, then a recursive call is made on the smaller subarray (of size $k-1$) and with the same i. Finally, if $k < i$, then a recursive call is made on the larger subarray (of size $n-k$) with i replaced by $i-k$. Deduce that the expected running time of Ms. Lucky's algorithm is $O(n \log n)$. (Notice that this running time may depend upon i (in addition to n). In your calculations, you may suitably ignore this dependence.) What is the worst-case running time of Ms. Lucky's algorithm?