1.2 Design a factor 2 approximation algorithm for the problem of finding a minimum cardinality maximal matching in an undirected graph.
Hint: Use the fact that any maximal matching is at least half the maximum matching.

1.3 (R. Bar-Yehuda) Consider the following factor 2 approximation algorithm for the cardinality vertex cover problem. Find a depth first search tree in the given graph, G, and output the set, say S, of all the nonleaf vertices of this tree. Show that S is indeed a vertex cover for G and $|S| \leq 2 \cdot \mathrm{OPT}$.
Hint: Show that G has a matching of size $\lceil |S|/2 \rceil$.

1.4 Perhaps the first strategy one tries when designing an algorithm for an optimization problem is the greedy strategy. For the vertex cover problem, this would involve iteratively picking a maximum degree vertex and removing it, together with edges incident at it, until there are no edges left. Show that this algorithm achieves an approximation guarantee of $O(\log n)$. Give a tight example for this algorithm.
Hint: The analysis is similar to that in Theorem 2.4.

1.5 A maximal matching can be found via a greedy algorithm: pick an edge, remove its two endpoints, and iterate until there are no edges left. Does this make Algorithm 1.2 a greedy algorithm?

1.6 Give a lower bounding scheme for the arbitrary cost version of the vertex cover problem.
Hint: Not easy if you don't use LP-duality.

1.7 Let $A = \{a_1, \ldots, a_n\}$ be a finite set, and let "$\leq$" be a relation on A that is reflexive, antisymmetric, and transitive. Such a relation is called a *partial ordering of* A. Two elements $a_i, a_j \in A$ are said to be *comparable* if $a_i \leq a_j$ or $a_j \leq a_i$. Two elements that are not comparable are said to be *incomparable*. A subset $S \subseteq A$ is a *chain* if its elements are pairwise comparable. If the elements of S are pairwise incomparable, then it is an *antichain*. A *chain (antichain) cover* is a collection of chains (antichains) that are pairwise disjoint and cover A. The size of such a cover is the number of chains (antichains) in it. Prove the following min–max result. The size of a longest chain equals the size of a smallest antichain cover.
Hint: Let the size of the longest chain be m. For $a \in A$, let $\phi(a)$ denote the size of the longest chain in which a is the smallest element. Now, consider the partition of A, $A_i = \{a \in A \mid \phi(a) = i\}$, for $1 \leq i \leq m$.

1.8 (Dilworth's theorem, see [202]) Prove that in any finite partial order, the size of a largest antichain equals the size of a smallest chain cover.
Hint: Derive from the König-Egerváry Theorem. Given a partial order on n-element set A, consider the bipartite graph $G = (U, V, E)$ with $|U| = |V| = n$ and $(u_i, v_j) \in E$ iff $a_i \leq a_j$.

The next ten exercises are based on Appendix A.

1.9 Is the following an **NP**-optimization problem? Given an undirected graph $G = (V, E)$, a cost function on vertices $c : V \rightarrow \mathbf{Q}^+$, and a positive integer k, find a minimum cost vertex cover for G containing at most k vertices.
Hint: Can valid instances be recognized in polynomial time (such an instance must have at least one feasible solution)?

1.10 Let $\mathcal{A}$ be an algorithm for a minimization **NP**-optimization problem Π such that the expected cost of the solution produced by $\mathcal{A}$ is $\leq \alpha\mathrm{OPT}$, for a constant $\alpha > 1$. What is the best approximation guarantee you can establish for Π using algorithm $\mathcal{A}$?
Hint: A guarantee of $2\alpha - 1$ follows easily. For guarantees arbitrarily close to α, run the algorithm polynomially many times and pick the best solution. Apply Chernoff's bound.

1.11 Show that if SAT has been proven **NP**-hard, and SAT has been reduced, via a polynomial time reduction, to the decision version of vertex cover, then the latter is also **NP**-hard.
Hint: Show that the composition of two polynomial time reductions is also a polynomial time reduction.

1.12 Show that if the vertex cover problem is in co-**NP**, then **NP** = co-**NP**.

1.13 (Pratt [230]) Let L be the language consisting of all prime numbers. Show that $L \in \mathbf{NP}$.
Hint: Consider the multiplicative group mod$n$, $Z_n^* = \{a \in \mathbf{Z}^+ \mid 1 \leq a < n \text{ and } (a, n) = 1\}$. Clearly, $|Z_n^*| \leq n - 1$. Use the fact that $|Z_n^*| = n - 1$ iff n is prime, and that Z_n^* is cyclic if n is prime. The Yes certificate consists of a primitive root of Z_n^*, the prime factorization of $n - 1$, and, recursively, similar information about each prime factor of $n - 1$.

1.14 Give proofs of self-reducibility for the optimization problems discussed later in this book, in particular, maximum matching, MAX-SAT (Problem 16.1), clique (Problem 29.15), shortest superstring (Problem 2.9), and Minimum makespan scheduling (Problem 10.1).
Hint: For clique, consider two possibilities, that v is or isn't in the optimal clique. Correspondingly, either restrict G to v and its neighbors, or remove v from G. For shortest superstring, remove two strings and replace them by a legal overlap (may even be a simple concatenation). If the length of the optimal superstring remains unchanged, work with this smaller instance. Generalize the scheduling problem a bit – assume that you are also given the number of time units already scheduled on each machine as part of the instance.

list. Let b_i and e_i denote the index of the first and last string in the ith group ($b_i = e_i$ is allowed). Thus, $b_1 = 1$. Let e_1 be the largest index of a string that overlaps with s_1 (there exists at least one such string, namely s_1 itself). In general, if $e_i < n$ we set $b_{i+1} = e_i + 1$ and denote by e_{i+1} the largest index of a string that overlaps with $s_{b_{i+1}}$. Eventually, we will get $e_t = n$ for some $t \le n$.

For each pair of strings (s_{b_i}, s_{e_i}), let $k_i > 0$ be the length of the overlap between their leftmost occurrences in s (this may be different from their maximum overlap). Let $\pi_i = \sigma_{b_i e_i k_i}$. Clearly, $\{\mathrm{set}(\pi_i) | 1 \le i \le t\}$ is a solution for $\mathcal{S}$, of cost $\sum_i |\pi_i|$.

The critical observation is that π_i does not overlap π_{i+2}. We will prove this claim for $i = 1$; the same argument applies to an arbitrary i. Assume, for contradiction, that π_1 overlaps π_3. Then the occurrence of s_{b_3} in s overlaps the occurrence of s_{e_1}. However, s_{b_3} does not overlap s_{b_2} (otherwise, s_{b_3} would have been put in the second group). This implies that s_{e_1} ends later than s_{b_2}, contradicting the property of endings of strings established earlier.

Because of this observation, each symbol of s is covered by at most two of the π_i's. Hence $\mathrm{OPT}_{\mathcal{S}} \le \sum_i |\pi_i| \le 2 \cdot \mathrm{OPT}$. □

The size of the universal set in the set cover instance $\mathcal{S}$ is n, the number of strings in the given shortest superstring instance. This fact, Lemma 2.11, and Theorem 2.4 immediately give the following theorem.

Theorem 2.12 *Algorithm 2.10 is a $2H_n$ factor algorithm for the shortest superstring problem, where n is the number of strings in the given instance.*

2.4 Exercises

2.1 Given an undirected graph $G = (V, E)$, the *cardinality maximum cut problem* asks for a partition of V into sets S and $\overline{S}$ so that the number of edges running between these sets is maximized. Consider the following greedy algorithm for this problem. Here v_1 and v_2 are arbitrary vertices in G, and for $A \subset V$, $d(v, A)$ denotes the number of edges running between vertex v and set A.

Algorithm 2.13

1. Initialization:
   $A \leftarrow \{v_1\}$
   $B \leftarrow \{v_2\}$
2. For $v \in V - \{v_1, v_2\}$ do:
   if $d(v, A) \ge d(v, B)$ then $B \leftarrow B \cup \{v\}$,
   else $A \leftarrow A \cup \{v\}$.
3. Output A and B.

Show that this is a factor 1/2 approximation algorithm and give a tight example. What is the upper bound on OPT that you are using? Give examples of graphs for which this upper bound is as bad as twice OPT. Generalize the problem and the algorithm to weighted graphs.

2.2 Consider the following algorithm for the maximum cut problem, based on the technique of *local search*. Given a partition of V into sets, the basic step of the algorithm, called *flip*, is that of moving a vertex from one side of the partition to the other. The following algorithm finds a *locally optimal solution* under the flip operation, i.e., a solution which cannot be improved by a single flip.

The algorithm starts with an arbitrary partition of V. While there is a vertex such that flipping it increases the size of the cut, the algorithm flips such a vertex. (Observe that a vertex qualifies for a flip if it has more neighbors in its own partition than in the other side.) The algorithm terminates when no vertex qualifies for a flip. Show that this algorithm terminates in polynomial time, and achieves an approximation guarantee of 1/2.

2.3 Consider the following generalization of the maximum cut problem.

Problem 2.14 (MAX k-CUT) Given an undirected graph $G = (V, E)$ with nonnegative edge costs, and an integer k, find a partition of V into sets $S_1, \ldots, S_k$ so that the total cost of edges running between these sets is maximized.

Give a greedy algorithm for this problem that achieves a factor of $(1 - \frac{1}{k})$. Is the analysis of your algorithm tight?

2.4 Give a greedy algorithm for the following problem achieving an approximation guarantee of factor 1/4.

Problem 2.15 (Maximum directed cut) Given a directed graph $G = (V, E)$ with nonnegative edge costs, find a subset $S \subset V$ so as to maximize the total cost of edges out of S, i.e., $\mathrm{cost}(\{(u \to v) \mid u \in S \text{ and } v \in \overline{S}\})$.

2.5 (N. Vishnoi) Use the algorithm in Exercise 2.2 and the fact that the vertex cover problem is polynomial time solvable for bipartite graphs to give a factor $\lceil \log_2 \Delta \rceil$ algorithm for vertex cover, where Δ is the degree of the vertex having highest degree.
Hint: Let H denote the subgraph consisting of edges in the maximum cut found by Algorithm 2.13. Clearly, H is bipartite, and for any vertex v, $\deg_H(v) \ge (1/2)\deg_G(v)$.

2.6 (Wigderson [265]) Consider the following problem.

Problem 2.16 (Vertex coloring) Given an undirected graph $G = (V, E)$, color its vertices with the minimum number of colors so that the two endpoints of each edge receive distinct colors.

1. Give a greedy algorithm for coloring G with $\Delta+1$ colors, where Δ is the maximum degree of a vertex in G.
2. Give an algorithm for coloring a 3-colorable graph with $O(\sqrt{n})$ colors.
   Hint: For any vertex v, the induced subgraph on its neighbors, $N(v)$, is bipartite, and hence optimally colorable. If v has degree $> \sqrt{n}$, color $v \cup N(v)$ using 3 distinct colors. Continue until every vertex has degree $\leq \sqrt{n}$. Then use the algorithm in the first part.

2.7 Let 2SC denote the restriction of set cover to instances having $f = 2$. Show that 2SC is equivalent to the vertex cover problem, with arbitrary costs, under approximation factor preserving reductions.

2.8 Prove that Algorithm 2.2 achieves an approximation factor of H_k, where k is the cardinality of the largest specified subset of U.

2.9 Give a greedy algorithm that achieves an approximation guarantee of H_n for set multicover, which is a generalization of set cover in which an integral coverage requirement is also specified for each element and sets can be picked multiple numbers of times to satisfy all coverage requirements. Assume that the cost of picking α copies of set S_i is $\alpha \cdot \text{cost}(S_i)$.

2.10 By giving an appropriate tight example, show that the analysis of Algorithm 2.2 cannot be improved even for the cardinality set cover problem, i.e., if all specified sets have unit cost.
Hint: Consider running the greedy algorithm on a vertex cover instance.

2.11 Consider the following algorithm for the weighted vertex cover problem. For each vertex v, $t(v)$ is initialized to its weight, and when $t(v)$ drops to 0, v is picked in the cover. $c(e)$ is the amount charged to edge e.

Algorithm 2.17

1. Initialization:
   $C \leftarrow \emptyset$
   $\forall v \in V,\ t(v) \leftarrow w(v)$
   $\forall e \in E,\ c(e) \leftarrow 0$
2. While C is not a vertex cover do:
   Pick an uncovered edge, say (u, v). Let $m = \min(t(u), t(v))$.
   $t(u) \leftarrow t(u) - m$
   $t(v) \leftarrow t(v) - m$
   $c(u, v) \leftarrow m$
   Include in C all vertices having $t(v) = 0$.
3. Output C.

Show that this is a factor 2 approximation algorithm.
Hint: Show that the total amount charged to edges is a lower bound on OPT and that the weight of cover C is at most twice the total amount charged to edges.

2.12 Consider the layering algorithm for vertex cover. Another weight function for which we have a factor 2 approximation algorithm is the constant function – by simply using the factor 2 algorithm for the cardinality vertex cover problem. Can layering be made to work by using this function instead of the degree-weighted function?

2.13 Use layering to get a factor f approximation algorithm for set cover, where f is the frequency of the most frequent element. Provide a tight example for this algorithm.

2.14 A *tournament* is a directed graph $G = (V, E)$, such that for each pair of vertices, $u, v \in V$, exactly one of (u, v) and (v, u) is in E. A *feedback vertex set* for G is a subset of the vertices of G whose removal leaves an acyclic graph. Give a factor 3 algorithm for the problem of finding a minimum feedback vertex set in a directed graph.
Hint: Show that it is sufficient to "kill" all length 3 cycles. Use the factor f set cover algorithm.

2.15 (Hochbaum [132]) Consider the following problem.

Problem 2.18 (Maximum coverage) Given a universal set U of n elements, with nonnegative weights specified, a collection of subsets of U, $S_1, \ldots, S_l$, and an integer k, pick k sets so as to maximize the weight of elements covered.

Show that the obvious algorithm, of greedily picking the best set in each iteration until k sets are picked, achieves an approximation factor of

$$1 - \left(1 - \frac{1}{k}\right)^k > 1 - \frac{1}{e}.$$

2.16 Using set cover, obtain approximation algorithms for the following variants of the shortest superstring problem (here s^R is the reverse of string s):

1. Find the shortest string that contains, for each string $s_i \in S$, both s_i and s_i^R as substrings.
   Hint: The universal set for the set cover instance will contain $2n$ elements, s_i and s_i^R, for $1 \leq i \leq n$.
2. Find the shortest string that contains, for each string $s_i \in S$, either s_i or s_i^R as a substring.
   Hint: Define $\text{set}(\pi) = \{s \in S \mid s \text{ or } s^R \text{ is a substring of } \pi\}$. Choose the strings π appropriately.