1. Consider the following statement.

   - Prove that every tree with n nodes has $n-1$ edges.

   A student in Cornfield University, learning induction from some badly-written template, sketch the following "proof":

   > When $n = 1$, the statement is clearly true. Assume that for $k \leq n$ the statement is true for any tree with k nodes. Let T be a tree with n nodes. By induction hypothesis T has $n-1$ edges. If we add one more node as a leaf to T and get a new tree T', the new tree T' has $n+1$ nodes and n edges. Therefore the statement is true for all n by induction.

   (a) Explain why the above proof is wrong.

   (b) Prove the statement correctly.

2. A *tournament* is a directed graph with exactly one directed edge between each pair of vertices. That is, for any vertices v and w, a tournament contains either an edge $v \to w$ or an edge $w \to v$, but not both. A *Hamiltonian path* in a directed graph G is a directed path that visits every vertex of G exactly once.

   Prove that every tournament contains a Hamiltonian path.

3. Consider the following sequence a_n: Set $a_1 = 2$, $a_2 = 3$, and define

   $$a_n = 1 + \min_{1<k<n} k \cdot a_{\lfloor \frac{n-1}{k} \rfloor} \qquad \text{for all } n \geq 3.$$

   Prove that $a_n \geq n$ for all positive integer n.

Here are some extra problems for you:

α. For any non-negative integer n, the $2^n \times 2^n$ checkerboard with an arbitrary block removed can be tiled using copies of the L-shaped triominos.

β. (a) Prove that every nonnegative integer n can be written as a sum of powers of 2. ("Write n in binary" is not a proof!)

(b) Prove that every nonnegative integer can be written as the sum of distinct, non-consecutive Fibonacci numbers. That is, if the Fibonacci number F_i appears in the sum, it appears exactly once, and its neighbors F_{i-1} and F_{i+1} do not appear at all. For example, $54 = F_9 + F_7 + F_5 + F_3$.

γ. Prove the *AM-GM inequality* using induction:

$$\frac{x_1 + \cdots + x_n}{n} \geq \sqrt[n]{x_1 \cdots x_n}.$$

[Hint: Handle the case for $n = 2$ separately. Try to prove the case when n is a power of 2 first.]

1. Let $G = (V, E)$ an undirected graph, with $|V| = n$ and $|E| = m$. Consider the following questions:
   (a) Describe an algorithm that finds a cycle in G.
   (b) Describe an algorithm that finds 2 distinct cycles in G. What complexity does your algorithm have? Can you do better? In particular, can you think of a way to find the 2 cycles in $O(n)$ time?
   (c) Describe an algorithm that checks if a graph has odd length cycles. If it does your algorithm should return no, otherwise it should return yes. What complexity does your algorithm have? What graph family does the above criterion characterize? Explain your answer.
2. Let G be a directed acyclic graph. Prove that G has a unique topological sort if and only if it has a Hamiltonian path.
3. Let G be an undirected graph, and let v and w be two of its vertices. Give an algorithm that computes the number of shortest $v - w$ paths in G in $O(m + n)$ time.

1. Let G be a directed graph with non-negative edge lengths. Describe an algorithm that given G and a number $\lambda > 0$ decides whether G has a cycle of average length strictly less than λ. The average length of a cycle is defined as the total length of the cycle (w/r/t the given edge lengths) divided by the number of edges in the cycle.

   Hint: Use negative cycle detection.

2. Let G be a graph and suppose two coins are placed on two of its vertices. In each step, both coins move to an adjacent node. A coin cannot remain in place on a step.

   a. Describe an algorithm that finds the shortest sequence of steps that puts the coins on the same node.

   b. Suppose both coins start facing heads, and that on every odd step we flip over the first coin and on every even step we flip over the second coin. Describe an algorithm that finds the shortest sequence of steps that puts the coins on the same node with both coins showing heads or both showing tails.

3. Let G be a directed graph and A a subset of its vertices. The distance from A to a vertex v is the shortest distance from any vertex $a \in A$ to v.

   Find the furthest vertex v from A along with the corresponding distance. (Alternatively, if G is a city map and A the set of fire stations, what is the furthest house from any fire station (and how far away)?)

4. Let G be a directed graph with weighted edges and H a subgraph of G with $V(H) = V(G)$. Let s and t be two vertices in G.

   Describe an algorithm that finds the shortest path from s to t using at most one edge not in H, and reports the extra edge not in H if one is used.

5. Let G be a directed graph. Each edge $e \in E$ has a probability $p(e) \in [0, 1]$ of failing. The edges fail independently of each other. The survival-rate of a path is the probability that no edge on it fails.

   Given two nodes s, t the goal is to find an s-t path P with the highest probability of surviving.

1. SOME RECURRENCES:

   (a) $t(0) = t(1) = 2$
   $t(n) = t\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + t\left(\left\lceil \frac{n}{2} \right\rceil\right) + 2n + 2$

   (b) $t(0) = 2$
   $t(1) = 3$
   $t(n) = 5 + t\left(\left\lfloor \frac{n}{2} \right\rfloor\right)$

   (c) $t(n) = \begin{cases} 2^n & \text{if } n < 2112014 \\ t\left(\left\lceil \frac{n}{2} \right\rceil\right) + 3 & \text{if } n \geq 2112014 \end{cases}$

2. COUNTING INVERSIONS: Let A be an array of n distinct numbers. An *inversion* in A is a pair of indices $i < j$ such that $A[j] < A[i]$; in English, it is a pair of elements out of increasing order.

   Describe an algorithm that computes the number of inversions in A in $O(n \log n)$ time.

3. Given a sorted array $A[1, \ldots, n]$ of distinct integers, you want to find out whether there is an index i for which $A[i] = i$. Give an algorithm that runs in time $O(\log n)$.

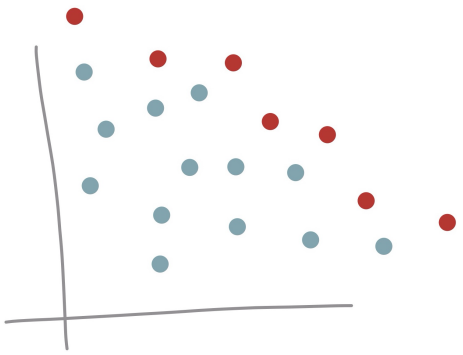
   [Hint: divide thy enemy, and then conquer thy enemy.]

4. Let A, B be two sorted arrays each of size n. Give an $O(\log n)$ algorithm to find the median element in the union of the two lists.

   [Hint: Suppose n is a power of 2 and try to find the nth element.]

5. Suppose we are given a set S of n points in the plane. For two points (u, v) and (x, y), we say that (x, y) *dominates* (u, v) if $x \geq u$ and $y \geq v$. A point p in S is *maximal* if there is no other point $q \in S$ dominating p.

   Describe an algorithm (the faster, the better) that computes the set of maximal points in S. For simplicity, you may assume that no two points in S have the same x or y-coordinate.



6. Suppose you are given the following tree traversals of a binary tree. Describe an algorithm that reconstructs a tree matching the traversals.

(a) **In**-order (For example: b d e f a c)
Pre-order (For example: e d b a f c)

(b) **In**-order (For example: b d e f a c)
Post-order (For example: b d f c a e)

(c) **Pre**-order (For example: e d b a f c)
Post-order (For example: b d f c a e)

1. A *subsequence* is anything obtained from a sequence by extracting a subset of elements, but keeping them in the same order; the elements of the subsequence need not be contiguous in the original sequence. For example, the strings DOG, DAMN, ROAM, YAM, RAMMING, CRAMMING, and DYNAMICPROGRAMMING are all subsequences of the sequence DYNAMICPROGRAMMING.

   Let $X[1...m]$ and $Y[1...n]$ be two arrays. A common subsequence of X and Y is another sequence that is a subsequence of both X and Y. Describe an efficient algorithm to compute the length of the longest common subsequence of X and Y.

2. Let $X[1..m]$ and $Y[1..n]$ be two arrays. A common *supersequence* of X and Y is another sequence to which X and Y are both subsequences. Describe an efficient algorithm to compute the length of the shortest common supersequence of X and Y.

3. Given a graph $G = (V, E)$, a vertex cover of G is a subset $S \subseteq V$ of vertices such that, for each edge $e = uv$ in G, u or v is in S. That is, the vertices in S cover all the edges. It is known that finding the minimum size vertex cover is NP-Hard in general graphs but it can be solved in trees using dynamic programming.

   This is the goal of this problem. Given a tree $T = (V, E)$ and a non-negative weight $w(v)$ for each vertex $v \in V$, give an algorithm that computes the minimum weight vertex cover of T. It is helpful to root the tree.

4. You are given a rectangle piece of cloth with dimensions $X \times Y$, where X and Y are positive integers, and a list of n products that can be made using the cloth. For each product $i \in [1..n]$ you know that a rectangle of cloth of dimensions $a_i \times b_i$ is needed that the final selling price of the product is c_i. Assume the a_i, b_i, and c_i are positive integers. You have a machine that can cut any rectangular piece of cloth into two pieces either horizontally or vertically. Design an algorithm that determines the best return on the $X \times Y$ piece of cloth; that is, a strategy for cutting the cloth so that the products made from the resulting pieces give the maximum sum of selling prices. You are free to make as many copies of a given product as you wish, or none if desired.

5. Consider the problem of covering point by intervals. Specifically, assume that you are given a set P of n points on the real line and a set of m intervals F each specified by two points on the real line. Describe an algorithm to find the minimum-weight set of intervals that covers all the points of P.

   (As a bonus, find a simple greedy algorithm for the unweighted case and prove its correctness.)

1. Minimum Vertex Cover in Trees

   Let $\mathcal{G}$ be an unweighted graph. A vertex cover of $\mathcal{G}$ is a set S of vertices in $\mathcal{G}$ such that every edge in $\mathcal{G}$ is incident to at least one vertex in S (i.e., the vertices in S cover the edges in $\mathcal{G}$.)

   Describe a greedy algorithm that computes the vertex cover of $\mathcal{G}$ if $\mathcal{G}$ is a tree, and prove its correctness.

2. Maximum Independent Sets in Trees

   Let $\mathcal{G}$ be an unweighted graph. An *independent set* of $\mathcal{G}$ is a set S of vertices in $\mathcal{G}$ such that no two vertices in S are connected by an edge. Finding the maximum independent set in a general graph is considered very hard.

   Suppose $\mathcal{G}$ was a tree. Describe a greedy algorithm that computes the maximum-size independent set in a tree.

3. Covering points by intervals

   Consider the problem of covering points by intervals. Specifically, assume that you are given a set P of n points on the real line and a set of m intervals F each specified by two points on the real line. Two discussions ago we solved the *weighted* case for the minimum *weight* set of intervals covering of P.

   In the unweighted case, describe a greedy algorithm to find the minimum number of intervals needed to cover all the points of P.

4. Piercing Intervals

   Let X be a set of n closed intervals on the real line. A set P of points *pierces* X if every interval in X contains at least one point in P. Describe and analyze an efficient algorithm to compute the smallest set of points that stabs X.

5. Weighted Scheduling

   We have n jobs $J_1, J_2, \ldots, J_n$ which we need to schedule on a machine. Each job J_i has a processing time t_i and a weight w_i. A schedule for the machine is an ordering of the jobs. Given a schedule, let C_i denote the finishing time of job J_i. For example, if job J_j is the first job in the schedule, its finishing time C_j is equal to t_j ; if job J_j follows job J_i in the schedule, its finishing time C_j is equal to $C_i + t_j$. The weighted completion time of the schedule is $\sum_{i=1}^{n} w_i C_i$.

   (a) For the case when $w_i = 1$ for all i, show that choosing the shortest job first is optimal.

   (b) (Harder) Give an efficient algorithm that finds a schedule with minimum weighted completion time given arbitrary weights.

1. Shortest Bottleneck Path

   Let $\mathcal{G}$ be an undirected graph with distinct nonnegative edge lengths. The *bottleneck distance* of a path in $\mathcal{G}$ is the length of the longest edge in the path.

   Prove that the minimum spanning tree of $\mathcal{G}$ contains the shortest bottleneck path between every pair of points.

2. Fixing a Minimum Spanning Tree

   Let $\mathcal{G}$ be an undirected graph with distinct edge weights. Let T be a minimum spanning tree in $\mathcal{G}$.

   Suppose the weight of a single edge e is altered. e may or may not be in T. Describe an $O(m+n)$ time algorithm that computes the MST of $\mathcal{G}$ with the updated weight.

3. Random Cuts

   Let G be a graph. A *cut* of G is a partition (A, B) of the vertices (i.e., A and B are subsets of $V(G)$, $A \cap B = \emptyset$, and $A \cup B = V(G)$. The *size* of a cut (A, B) is the number of edges $(u, v) \in E(G)$ such that $u \in A$ and $v \in B$.

   Prove that the expected size of a random cut is $|E(G)|/2$. That is, suppose that each vertex is placed in A with probability $1/2$ and in B with probability $1/2$, and let X denote the size of the resulting cut. Prove that $\mathbb{E}[X] = |E(G)|/2$.

4. Bůruvka's Algorithm

   Bůruvka's algorithm computes the MST of a graph G as follows. At the beginning of the algorithm, we have a forest where each vertex in $V(G)$ is its own connected component. At each iteration, the algorithm goes through every connected component in the graph and picks the cheapest edge adjacent to it, adding it to the forest.

   (a) Show that Bůruvka's algorithm decreases the number of vertices by a factor of two at each iteration.

   (b) Conclude that Bůruvka's algorithm takes $O(m \log n)$ time in the worst case.

5. Nuts and Bolts

   Suppose we are given n nuts and n bolts of different sizes. Each nut matches exactly one bolt and vice versa. The nuts and bolts are all almost exactly the same size, so we can't tell whether one bolt is smaller than the other, or if one nut is bigger than the other. If we try to match a nut with a bolt, it is either too big, too small, or an exact match. Describe an algorithm that matches the all the nuts and bolts in $O(n \log n)$ time in expectation.

1. Describe and analyze an algorithm to find the second smallest spanning tree of a given graph $\mathcal{G}$; that is, the spanning tree of $\mathcal{G}$ with smallest total weight except for the minimum spanning tree.

2. Let S be a set of n points in the plane. A point p in S is called *maximal* if no other point in S is both above and to the right of p. If each point in s is chosen independently and uniformly at random from the unit square $[0,1]\times[0,1]]$, what is the exact expected number of maximal points in S?

3. A *data stream* is an extremely long sequence of items that you can only read in one pass (i.e., each item can only be read once).

   (a) Describe and analyze an algorithm that chooses one element uniformly at random from a data stream, without knowing the length of the stream in advance. Your algorithm should spend $O(1)$ time per stream item and use $O(1)$ space, not including the stream itself (i.e., the space requirement should not depend on the length of the stream).

   (b) Describe and analyze an algorithm that chooses k elements uniformly at random from a data stream, without knowing the length of the stream in advance. Again, your algorithm should spend $O(1)$ time per stream item and use $O(k)$ space, not including the stream itself.

4. Consider the Minimum Spanning Tree Problem on an undirected graph. $G = (V, E)$, with a cost $c_e \geq 0$ on each edge, where the costs may not all be different. If the costs are not all distinct, there can in general be many distinct minimum-cost solutions. Suppose we are given a spanning tree T with the guarantee that for every $e \in T$, e belongs to some minimum-cost spanning tree in G. Can we conclude that T itself must be a minimum-cost spanning tree in G? Give a proof or a counterexample with explanation.

1. Suppose you are given a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, two vertices s and t, a capacity function $c : \mathcal{E} \to \mathbb{R}_{>0}$, and a second function $f : \mathcal{E} \to \mathbb{R}$. Describe an algorithm to determine whether f is a maximum (s,t)-flow in $\mathcal{G}$.

2. In a standard $s-t$ maximum flow problem, we assume that edges have capacities, and there is no limit on how much flow is allowed to pass through a node. In this problem, we consider the variant where nodes have capacities.

   Let $G = (V, E)$ be a directed graph with source s and sink t. Let $c : V \to \mathbb{R}^+$ be a capacity function. Recall that a flow f assigns a flow value $f(e)$ to each edge e. A flow f is feasible if the total flow into every vertex v is at most $c(v)$:

   $$f^{in}(v) \leq c(v) \quad \forall v.$$

   Describe a reduction from the problem of finding a feasible $s-t$ flow of maximum value in G to the standard flow problem with edge capacities.

3. Let $G = (V, E)$ be a directed graph with non-negative edge-capacities $c : E \to \mathbb{R}^+$. For any two distinct nodes $a, b \in V$, let $\alpha_G(a, b)$ be the capacity of the minimum cut between a and b in G. For any three distinct nodes $u, v, w \in V$ prove the following:

   $$\alpha_G(u, v) \geq \min\{\alpha_G(u, w), \alpha_G(w, v)\}$$

4. Given an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with three vertices u, v, and w, describe and analyze an algorithm to determine whether there is a path from u to w that passes through v.

1. Let (G, s, t) be a flow network with integer capacities. An edge e is *upper-binding* if increasing the capacity on e increases the value of the maximum flow in G.

   Describe and analyze an algorithm that finds all the upper-binding edges in the graph.

2. The NSA has established several monitoring stations around the country, each one conveniently hidden in the back of a Starbucks. Each station can monitor up to 42 cell-phone towers, but can only monitor cell-phone towers within a 20-mile radius. To ensure that every cell-phone call is recorded even if some stations malfunction, the NSA requires each cell-phone tower to be monitored by at least 3 different stations.

   Suppose you know that there are n cell-phone towers and m monitoring stations, and you are given a function $\textsc{Distance}(i, j)$ that returns the distance between the i-th tower and the j-th station in $O(1)$ time. Describe and analyze an algorithm that either computes a valid assignment of cell-phone towers to monitoring stations, or reports correctly that there is no such assignment (in which case the NSA will build yet another Starbucks).

3. Let G = (V, E) be an arbitrary connected graph with weighted edges.

   (a) Prove that for any partition of the vertices V into two disjoint subsets, the minimum spanning tree of G includes a minimum-weight edge among those with one endpoint in each subset.

   (b) Prove that for any cycle in G, the minimum spanning tree of G excludes the maximum-weight edge in that cycle.

4. A **polygonal path** is a sequence of line segments joined end-to-end; the endpoints of these line segments are called the **vertices** of the path. The **length** of a polygonal path is the sum of the lengths of its segments. A polygonal path with vertices $(x_1, y_1), (x_2, y_2), ..., (x_k, y_k)$ is **monotonically increasing** if $x_i < x_{i+1}$ and $y_i < y_{i+1}$ for every index i – informally, each vertex of the path is above and to the right of its predecessor.

   Suppose you are given a set S of n points in the plane, represented as two arrays $X[1...n]$ and $Y[1...n]$. Describe and analyze an algorithm to compute the length of the maximum-length monotonically increasing path with vertices in S.

5. Suppose we are given an n-digit integer X. Repeatedly remove one digit from either end of X (your choice) until no digits are left. The *square-depth* of X is the maximum number of perfect squares that you can see during this process.

   Describe and analyze an algorithm to compute the square-depth of a given integer X, represented as an array $X[1...n]$ of n decimal digits. Assume you have access to a subroutine $\textsc{IsSquare}$ that determines whether a given k-digit number is a perfect square in $O(k^2)$ time.

6. Consider a full binary tree of height h. Starting from the root, flip a coin and go the left subtree with probability 1/2 (if you get a head), and to the right subtree with

probability 1/2 (if you get a tail). Continue to flip coins and take subtrees until you arrive at a leaf.

Define the random variable

$$X_h = \alpha^k,$$

where k is the number of left subtrees taken on the walk down to the leaf. Prove that $E[X_h] = (\frac{1+\alpha}{2})^h$.

1. Suppose we are given an array $A[1..m][1..n]$ of non-negative real numbers such that each row and column sum is an integer. We want to round A to an integer matrix, replacing each entry x in A with either $\lceil x \rceil$ or $\lfloor x \rfloor$ while maintaining the sum of entries in any row or column of A. For example,

$$\begin{pmatrix} 1.2 & 3.4 & 2.4 \\ 3.9 & 4.0 & 2.1 \\ 7.9 & 1.6 & 0.5 \end{pmatrix} \text{ rounds to } \begin{pmatrix} 1 & 4 & 2 \\ 4 & 4 & 2 \\ 8 & 1 & 1 \end{pmatrix}.$$

   Describe an algorithm that either outputs a feasible rounding scheme or outputs that there isn't one.

2. A group of n people $p_1, ..., p_n$ are trying to figure out a schedule over the next n nights $d_1, ..., d_n$ such that each person cooks exactly once. For each person p_i, there is a set of nights S_i that the person is not able to cook.

   A feasible dinner schedule is an assignment of each peorson to a different night such that each person cooks on exactly one night, there is someone cooking on each night, and if p_i cooks on night d_j, then $d_j \notin S_i$.

   (a) Describe a bipartite graph G such that g has a perfect matching if and only if there is a feasible dinner schedule for the group.

   (b) Suppose we already have at erroneous schedule in which two people p_i and p_j have been assigned to cook on the same day d_l, while no one has been assigned to d_k, and everyone else is assigned correctly. Describe an algorithm that, in $O(n^2)$ time, decides whether or not there exists a feasible dinner schedule, and outputs a feasible schedule if one exists.

3. In the min-cost flow problem, we are given a graph G and have to send exactly k units of flow from s to t, each edge e has a capacity $c(e)$ and a cost $w(e)$ associated with it, and we want to minimize the cost of the flow, where the cost of a flow f is $\sum_{e \in E(G)} w(e)f(e)$. (The flow f must satisfy conservation and capacity constraints.)

   Assume that you have a black box that can solve the min-cost flow problem in polynomial time.

   (a) Describe how to compute k edge disjoint paths from s and t such that the total cost of these paths is minimized.

   (b) Suppose G is bipartite. Describe an algorithm that decides if this graph has a perfect matching, and, if so, outputs the cheapest such matching.

   (c) Banana has just released a new version of their iFifi – the first electronic gizmo that is simultaneously dishwasher-safe and available in two colors: black and midnight.

   Banana has k distribution centers $C_1, ..., C_k$, and center C_i has t_i iFifis in stock. You need to plan the distribution of the iFifis to the Banana stores. You have a list of n stores $S_1, ..., S_n$, and for each one of them, there is a quota f_i of how

many iFifis they need. For every distribution center C_i and store S_j, you know the distance between them in miles (rounded up to an integer).

Sending a single iFifi from a distribution center C_i to a store S_i costs d_{ij} dollars. Describe an algorithm that computes the minimum cost way to send all the required iFifis from the centers to the stores.

4. Let $G = (V, E)$ be an undirected graph. A *dominating set* $S \subset V$ is a subset of vertices such that every vertex $v \in V$ is either in S or adjacent to a vertex in S. The *minimum dominating set* is a dominating set of the smallest cardinality among all dominating sets in G.

   Reduce minimum dominating set and minimum set cover to one another, in both directions.

5. A subset S of vertices in an undirected graph G is called *triangle-free* if the induced graph G_S has no 3-cycles (aka triangles). That is, for every three vertices $u, v, w \in V$, at least one of the three edges uv, uw, vw is absent from G.

   Prove that finding the size of the largest triangle-free subset is NP-hard.

1. From Set Cover to Montone SAT

   Consider an instance ϕ of a CNF formula specified by clauses $C_1, C_2, \ldots, C_k$ over a set of boolean variables $x_1, x_2, \ldots, x_n$. We say ϕ is *monotone* if each term in each clause consists of a nonnegated variable; that is, each term is equal to x_i and not $\bar{x_i}$ - negations are not allowed. They could be easily satisfied by setting each variable to 1.

   For example, the clause

   $$\phi(x_1, x_2, x_3) = (x_1 \vee x_2) \wedge (x_1 \vee x_3) \wedge (x_3 \vee x_2)$$

   could be satisfied by setting all thre variables to 1, or by setting x_1 and x_2 to 1 and x_3 to 0.

   Given a monotone instance ϕ of a CNF formula and an integer $k \in \mathbb{N}$, the *Monotone Satisfiability* problem asks whether there is a satisfying assignment for the instance in which at most k variables are set to 1.

   The *Set Cover* decision problem asks, given a collection F of subsets $S_1, S_2, \ldots, S_m$ of a ground set $U = \{1, \ldots, n\}$, and an integer k, whether U can be covered by k sets in F (i.e., whether there are k sets in F whose union is U).

   Give a Karp reduction from the Set Cover decision problem to Monotone Satisfiability.

2. Reducing from 3-Coloring to SAT

   The *3-Coloring* problem asks whether the vertices of a given graph G can be colored by three colors such that any two adjacent vertices have different colors. This problem is NP-Complete.

   (a) The 42-Coloring coloring problem asks, given a graph G, whether the vertices of G can be colored by 42 different colors such that any two adjacent vertices are colored differently. Prove that the 42-Coloring problem is NP-Complete. *[Hint: Reduce from 3-Coloring.]*

   (b) The 42-SAT problem asks, given a boolean formula ϕ in conjunctive normal form with exactly 42 literals per clause, determine whether ϕ has a satisfying assignment. Prove that the 42-SAT problem is NP-Complete. *[Hint: Reduce from 3SAT.]*

3. Self-Reduction for k-Coloring

   In the *k-Coloring* problem, we are given a graph G and an integer k and have to decide whether the vertices of G can be colored by k-different colors such that no two adjacent vertices are colored the same.

   Suppose we have a black-box that can answer the k-Coloring decision problem (where k is fixed). Design an algorithm that uses only a polynomial number of calls to the black-box that either generates a k-Coloring of a graph or determines that a k-Coloring does not exist.

4. In the Hitting Set Problem, we are given a collection C of subsets of a finite set S. That is, $C = \{C_1, ..., C_n\}$, where for each i we have $C_i \subseteq S$. We are also given a parameter k, and we want know whether there exists a subset S' of S such that S' contains at least one element from each C_i and $|S'| = k$. Prove that this problem is NP-Complete.

1. Let $G = (V, E)$ by a directed graph. A *half-of-a-Hamiltonian-path* is a path P that visits $n/2$ vertices, where $n = |V|$ is the total number of vertices in G. Prove that deciding whether G contains a half-of-a-Hamiltonian-path is NP-complete.

2. Let G be a directed, weighted graph. Given such a G, the zero-length cycle problem asks us to check if G has a simple cycle C such that the sum of the weights on the edges in C is exactly 0. Prove that this problem is NP-Complete.

3. In the *Bounded Degree Spanning Tree* problem we are given a graph G and a number k. It should return "yes" exactly when the graph has a spanning tree where each vertex has degree at most k. Prove that this problem is NP-complete.

4. In the *subset sum problem*, you are given n positive integers $X = \{x_1, ..., x_k\}$ and a positive integer S, and you want to know whether there exists a subset Y of X that sums to S.

   Assume you have a black box that answers the decision version of this problem. Use a polynomial number of calls to the black box to construct such a Y, if one exists.

5. In the 2-partition problem, we are given positive integers $A = \{a_1, ..., a_n\}$, and we want to know whether there is a partition of the numbers into two sets such that the sum of the numbers in each set is exactly $(\Sigma_i a_i)/2$. Prove that this problem is NP-Complete.

6. A *Canadian graph* $G = (V, E)$ is a directed graph such that each edge is colored red or white. A *Canadian path* in a Canadian graph G is a path P whose edges alternate between red and white; that is, no two consecutive edges in P are both red or both white. A *Canadian Hamiltonian path* in G is a path that is both Canadian and Hamiltonian path.

   (a) Prove that the problem of deciding whether G contains a Canadian Hamiltonian path is NP-complete.

   (b) (Harder) In the opposite direction, reduce the Canadian Hamiltonian path decision problem to the Hamiltonian path problem.

7. Two graphs are *isomorphic* if one can be transformed into the other by relabeling the vertices. Consider the following decision problems:

   - Graph Isomorphism: Given two graphs G and H, determine whether G and H are isomorphic.
   - Even Graph Isomorphism: Given two graphs G and H, such that every vertex in G and H has even degree, determine whether G and H are isomorphic.
   - Subgraph Isomorphism: Given two graphs G and H, determine whether G is isomorphic to a subgraph of H.

   (a) Describe a polynomial-time reduction from Graph Isomorphism to Even Graph Isomorphism.

(b) Describe a polynomial-time reduction from Graph Isomorphism to Subgraph Isomorphism.

(c) Prove that Subgraph Isomorphism is NP-Complete by reducing from Clique.

1. Pebbling is a solitaire game played on an undirected graph G, where each vertex has zero or more pebbles. A single pebbling move consists of removing two pebbles from a vertex v and adding one pebble to a neighbor of v (the vertex v must have ta least two pebbles before the move.)

   Given a graph $G = (V, E)$ and a pebble count $p(v)$ for each vertex v, the Pebble Destruction asks whether there is a sequence of pebbling moves that removes all but one pebble. Prove that this problem is NP-complete.

2. In the Multiple Interval Scheduling problem, each job requires a set of intervals of time during which it needs to use the processor. For example, a single job could require the processor from 10 AM to 11 AM, and again from 2 PM to 3 PM. If you accept this job, it ties up your processor during those two hours, but you could still accept jobs that need any other time periods, including the hours from 11 AM to 2 PM.

   For a given number k, we want to know if it is possible to accept at least k of the jobs so that no two of the accepted jobs have any overlap in time. Prove that this problem is NP-complete.

3. The ranking officer of the ROTC decides to postpone a picnic and must notify everyone else.

   Each ROTC person except the ranking officer reports to a single superior officer. If u is the superior officer of v, then v is the direct subordinate of u.

   To notify everyone of the postponement, the ranking officer first calls each of her direct subordinates, one at a time. As soon as each subordinate gets the phone call, he or she immediately notifies each of his or her direct subordinates, one at a time. The process continues this way until everyone has been notified. Note that in this process each person can only call direct subordinates.

   We can picture this process as being divided into rounds. At each round, each person who has already learned of the postponement can call exactly one of his or her direct subordinates on the phone.

   The number of rounds it takes for everyone to be notified depends on the sequence in which each person calls their direct subordinates. Give an efficient algorithm that determines the minimum number of rounds needed for everyone to be notified.

4. You are organizing a three-day event where several 30-minute sets of music will be played. You need to hire DJs according to the following constraints:

   - Exactly k sets of music must be played each day, and thus $3k$ sets altogether.
   - Each set must be played by a single DJ in a single music genre.
   - Each genre can only be played at most once per day.
   - Each candidate DJ has given you a list of genres they are willing to play.
   - Each DJ can play at most three sets during the entire event.

Suppose there are n DJs and g genres. Describe and analyze an efficient algorithm that either assigns a DJ and a genre to each of the $3k$ sets, or correctly reports that no such assignment is possible.

5. Suppose $G = (V, E)$ is a directed graph with edge weights that may be negative, but contains no negative cycles. Let $l(e)$ denote the length of edge e. A *price function* p is a function that assigns a real number to each vertex. Given any price function p, the *reduced cost function* l_p is:

$$l_p(u, v) = p(u) + l(u, v) - p(v).$$

   (a) Prove that every reduced cost function preserves negative cycles and shortest paths.

   (b) We say that a price function p is *feasible* if $l_p(u, v) \geq 0$ for all edges (u, v). Suppose there exists a vertex s that can reach every other vertex by a path, and define a price function p by letting $p(u)$ be the length of the shortest path from s to u. Prove that p is feasible.

   (c) Combine your two previous answers to show that computing shortest paths from k sources in a graph with negative weight edges can be accomplished with only one application of Bellman-Ford and $k - 1$ applications of Dijkstra.

6. Solve the following recurrences:

   (a) $T(n) = T(\lceil n/15 \rceil) + T(\lceil n/10 \rceil) + 2T(\lceil n/6 \rceil) + n$.

   (b) $T(n) = T(n-1) + 2n - 1, T(0) = 0$.

   (c) $T(n) = 5T(n/2) + n^2, T(1) = 1$.

   (d) $T(n) = 4T(n/2) + n^2, T(1) = 1$.

   (e) $T(n) = 3T(n/2) + n^2, T(1) = 1$.