CSCE 411 Design and Analysis of Algorithms
Project Exercises for Summer 2020

June 2020

Problem 1 (10 points):

A *contiguous subsequence* of a list S is a subsequence made up of consecutive elements of S. For instance, if S is

$$5, 15, -30, 10, -5, 40, 10,$$

then 15, -30, 10 is a contiguous subsequence but 5, 15, 40 is not. Give a linear-time algorithm for the following task:

Input: A list of numbers a_1, a_2, $\cdots$, a_n.

Output: The contiguous subsequence of maximum sum (a subsequence of length zero has sum zero).

For the preceding example, the answer would be 10, -5, 40, 10, with a sum of 55.

(Hint: For each $j \in \{1, 2, \cdots, n\}$, consider contiguous subsequences ending exactly at position j.)

Problem 2 (10 points):

Yuckdonald's is considering opening a series of restaurants along Quaint Valley Highway (QVH). The n possible locations are along a straight line, and the distances of these locations from the start of QVH are, in miles and in increasing order, m_1, m_2, $\cdots$, m_n. The constraints are as follows:

- At each location, Yuckdonald's may open at most one restaurant. The expected profit from opening a restaurant at location i is p_i, where $p_i > 0$ and $i = 1, 2, \cdots, n$.
- Any two restaurants should be at least k miles apart, where k is a positive integer.

Give an efficient algorithm to compute the maximum expected total profit subject to the given constraints.

Problem 3 (10 points):

You are given a string of n characters $s[1 \cdots n]$, which you believe to be a corrupted text document in which all punctuation has vanished (so that it looks something like "itwasthebestoftimes$\cdots$"). You wish to reconstruct the document using a dictionary, which is available in the form of a Boolean function dict$(\cdot)$: for any string w, dict(w) equals "true" if w is a valid word, and equals "false" otherwise.

- (a) Give a dynamic programming algorithm that determines whether the string $s[\cdot]$ can be reconstituted as a sequence of valid words. The running time should be at most $O(n^2)$, assuming calls to dict take unit time.
- (b) In the event that the string is valid, make your algorithm output the corresponding sequence of words.

Problem 4 (10 points):

Pebbling a checkerboard. We are given a checkerboard which has 4 rows and n columns, and has an integer written in each square. We are also given a set of $2n$ pebbles, and we want to place some or all of these on the checkerboard (each pebble can be placed on exactly one square) so as to maximize the sum of the integers in the squares that are covered by pebbles. There is one constraint: for a placement of pebbles to be legal, no two of them can be on horizontally or vertically adjacent squares (diagonal adjacency is fine).

(a) Determine the number of legal patterns that can occur in any column (in isolation, ignoring the pebbles in adjacent columns) and describe these patterns.

Call two patterns *compatible* if they can be placed on adjacent columns to form a legal placement. Let us consider subproblems consisting of the first k columns, for $1 \leq k \leq n$. Each subproblem can be assigned a *type*, which is the pattern occurring in the last column.

(b) Using the notions of compatibility and type, give an $O(n)$-time dynamic programming algorithm for computing an optimal placement.

Problem 5 (10 points):

A subsequence is *palindromic* if it is the same whether read left to right or right to left. For instance, the sequence

$$A, C, G, T, G, T, C, A, A, A, A, T, C, G$$

has many palindromic subsequences, including A, C, G, C, A and A, A, A, A (on the other hand, the subsequence A, C, T is not palindromic). Devise an algorithm that takes a sequence $x[1 \cdots n]$ and returns the (length of the) longest palindromic subsequence. Its running time should be $O(n^2)$.

Problem 6 (10 points):

You are given a convex polygon P on n vertices in the plane (specified by their x and y coordinates). A triangulation of P is a collection of $n-3$ diagonals

of P such that no two diagonals intersect (except possibly at their endpoints). Notice that a triangulation splits the polygon's interior into $n-2$ disjoint triangles. The *cost* of a triangulation is the sum of the lengths of the diagonals in it. Give an efficient algorithm for finding a triangulation of minimum cost. (*Hint*: Label the vertices of P by $1, \cdots, n$, starting from an arbitrary vertex and walking clockwise. For $1 \leq i < j \leq n$, let the subproblem $A(i, j)$ denote the minimum cost triangulation of the polygon spanned by vertices $i, i+1, \cdots, j$.)

Problem 7 (10 points):

Cutting cloth. You are given a rectangular 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 $i \in \{1, 2, \cdots, n\}$ you know that a rectangle of cloth of dimensions $a_i \times b_i$ is needed and that the final selling price of the product is c_i. Assume the a_i , b_i , and c_i are all 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.

Problem 8 (10 points):

A mission-critical production system has n stages that have to be performed sequentially; stage i is performed by machine M_i. Each machine M_i has a probability r_i of functioning reliably and a probability $1 - r_i$ of failing (and the failures are independent). Therefore, if we implement each stage with a single machine, the probability that the whole system works is $r_1 \cdot r_2 \cdots r_n$. To improve this probability we add *redundancy*, by having m_i copies of the machine M_i that performs stage i. The probability that all m_i copies fail simultaneously is only $(1-r_i)^{m_i}$, so the probability that stage i is completed correctly is $1-(1-r_i)^{m_i}$ and the probability that the whole system works is $\prod_{i=1}^{n}(1-(1-r_i)^{m_i})$. Each machine M_i has a cost c_i, and there is a total budget B to buy machines. (Assume that B and c_i are positive integers.) Given the probabilities r_1, $\cdots$, r_n, the costs c_1, $\cdots$, c_n, and the budget B, find the redundancies m_1, $\cdots$, m_n that are within the available budget and that maximize the probability that the system works correctly.

Problem 9 (10 points):

Prove the following two properties of the Huffman encoding scheme.

- (a) If some character occurs with frequency more than $\frac{2}{5}$, then there is guaranteed to be a codeword of length 1.
- (b) If all characters occur with frequency less than $\frac{1}{3}$, then there is guaranteed to be no codeword of length 1.

Problem 10 (10 points):

Under a Huffman encoding of n symbols with frequencies f_1, f_2, $\cdots$, f_n, what is the longest a codeword could possibly be? Give an example set of frequencies that would produce this case.

Problem 11 (10 points):

Give a linear-time algorithm that takes as input a tree and determines whether it has a perfect matching: a set of edges that touches each node exactly once.

Problem 12 (10 points):

Here's a problem that occurs in automatic program analysis. For a set of variables x_1, x_2, $\cdots$, x_n, you are given some *equality* constraints of the form "$x_i = x_j$" and some *disequality* constraints of the form "$x_i \neq x_j$". Is it possible to satisfy all of them?

For instance, the constraints

$$x_1 = x_2, \quad x_2 = x_3, \quad x_3 = x_4, \quad x_1 \neq x_4$$

cannot be satisfied. Give an efficient algorithm that takes as input m constraints over n variables and decides whether the constraints can be satisfied.

Problem 13 (10 points):

Graphs with prescribed degree sequences. Given a list of n positive integers d_1, d_2, $\cdots$, d_n, we want to efficiently determine whether there exists an undirected graph $G = (V, E)$ whose nodes have degrees precisely d_1, d_2, $\cdots$, d_n. That is, if $V = \{v_1, \cdots, v_n\}$, then the degree of v_i should be exactly d_i. We call $(d_1, \cdots, d_n)$ the *degree sequence* of G. This graph G should not contain self-loops (edges with both endpoints equal to the same node) or multiple edges between the same pair of nodes.

- (a) Give an example of d_1, d_2, d_3, d_4 where all the $d_i \leq 3$ and $d_1 + d_2 + d_3 + d_4$ is even, but for which no graph with degree sequence (d_1, d_2, d_3, d_4) exists.
- (b) Suppose that $d_1 \geq d_2 \geq \cdots \geq d_n$ and that there exists a graph $G = (V, E)$ with degree sequence $(d_1, \cdots, d_n)$. We want to show that there must exist a graph that has this degree sequence and where in addition the neighbors of v_1 are v_2, v_3, $\cdots$, v_{d_1+1}. The idea is to gradually transform G into a graph with the desired additional property.
  - i. Suppose the neighbors of v_1 in G are not v_2, v_3, $\cdots$, v_{d_1+1}. Show that there exists $i < j \leq n$ and $u \in V$ such that $(v_1, v_i), (u, v_j) \notin E$ and $(v_1, v_j), (u, v_i) \in E$.
  - ii. Specify the changes you would make to G to obtain a new graph $G' = (V, E')$ with the same degree sequence as G and where $(v_1, v_i) \in E'$.

  - iii. Now show that there must be a graph with the given degree sequence but in which v_1 has neighbors v_2, v_3, $\cdots$, v_{d_1+1}.

- (c) Using the result from part (b), describe an algorithm that on input d_1, $\cdots$, d_n (not necessarily sorted) decides whether there exists a graph with this degree sequence. Your algorithm should run in time polynomial in n.

Problem 14 (10 points):

The basic intuition behind Huffman's algorithm, that frequent blocks should have short encodings and infrequent blocks should have long encodings, is also at work in English, where typical words like "I", "you", "is", "and", "to", "from", and so on are short, and rarely used words like "velociraptor" are longer.

However, words like "fire!", "help!", and "run!" are short not because they are frequent, but perhaps because time is precious in situations where they are used.

To make things theoretical, suppose we have a file composed of m different words, with frequencies f_1, $\cdots$, f_m. Suppose also that for the i-th word, the cost per bit of encoding is c_i . Thus, if we find a prefix-free code where the i th word has a codeword of length l_i, then the total cost of the encoding will be $\sum_i f_i \cdot c_i \cdot l_i$.

Show how to find the prefix-free encoding of minimal total cost.

Problem 15 (10 points):

A server has n customers waiting to be served. The service time required by each customer is known in advance: it is t_i minutes for customer i. So if, for example, the customers are served in order of increasing i, then the i-th customer has to wait $\sum_{j=1}^{i} t_j$ minutes.

We wish to minimize the total waiting time

$$T = \sum_{i=1}^{n} (\text{time spent waiting by customer } i).$$

Give an efficient algorithm for computing the optimal order in which to process the customers.

Problem 16 (10 points):

In an undirected graph, the *degree* $d(u)$ of a vertex u is the number of neighbors u has, or equivalently, the number of edges incident upon it. In a directed graph, we distinguish between the *indegree* $d_{in}(u)$, which is the number of edges into u, and the *outdegree* $d_{out}(u)$, the number of edges leaving u.

- (a) Show that in an undirected graph, $\sum_{u \in V} d(u) = 2|E|$.

- Use part (a) to show that in an undirected graph, there must be an even number of vertices whose degree is odd.

- Does a similar statement hold for the number of vertices with odd indegree in a directed graph?

Problem 17 (10 points):
Design a linear-time algorithm which, given an undirected graph G and a particular edge e in it, determines whether G has a cycle containing e.

Problem 18 (10 points):
Undirected vs. directed connectivity.

- (a) Prove that in any connected undirected graph $G = (V, E)$ there is a vertex $v \in V$ whose removal leaves G connected. (Hint: Consider the DFS search tree for G.)
- (b) Give an example of a strongly connected directed graph $G = (V, E)$ such that, for every $v \in V$, removing v from G leaves a directed graph that is not strongly connected.
- (c) In an undirected graph with 2 connected components it is always possible to make the graph connected by adding only one edge. Give an example of a directed graph with two strongly connected components such that no addition of one edge can make the graph strongly connected.

Problem 19 (10 points):
Give an efficient algorithm that takes as input a directed acyclic graph $G = (V, E)$, and two vertices $s, t \in V$, and outputs the number of different directed paths from s to t in G.

Problem 20 (10 points):
Give a linear-time algorithm for the following task.
Input: A directed acyclic graph G
Question: Does G contain a directed path that touches every vertex exactly once?

Problem 21 (10 points):
Two paths in a graph are called *edge-disjoint* if they have no edges in common. Show that in any undirected graph, it is possible to pair up the vertices of odd degree and find paths between each such pair so that all these paths are edge-disjoint.

Problem 22 (10 points):
You are given a directed graph $G = (V, E)$ with (possibly negative) weighted edges, along with a specific node $s \in V$ and a tree $T = (V, E')$, with $E' \subseteq E$.

Give an algorithm that checks whether T is a shortest-path tree for G with starting point s. Your algorithm should run in linear time.

Problem 23 (10 points):

Give an algorithm that takes as input a directed graph with positive edge lengths, and returns the length of the shortest cycle in the graph (if the graph is acyclic, it should say so). Your algorithm should take time at most $O(|V|^3)$.

Problem 24 (10 points):

You are given a strongly connected directed graph $G = (V, E)$ with positive edge weights along with a particular node $v_0 \in V$. Give an efficient algorithm for finding shortest paths between all pairs of nodes, with the one restriction that these paths must all pass through v_0.

Problem 25 (10 points):

Generalized shortest-paths problem. In Internet routing, there are delays on lines but also, more significantly, delays at routers. This motivates a generalized shortest-paths problem.

Suppose that in addition to having edge lengths $\{l_e : e \in E\}$, a graph also has *vertex costs* $\{c_v : v \in V\}$. Now define the cost of a path to be the sum of its edge lengths, plus the costs of all vertices on the path (including the endpoints). Give an efficient algorithm for the following problem.

Input: A directed graph $G = (V, E)$; positive edge lengths l_e and positive vertex costs c_v; a starting vertex $s \in V$.

Output: An array cost$[\cdot]$ such that for every vertex u, cost$[u]$ is the least cost of any path from s to u (i.e., the cost of the cheapest path), under the definition above.

Notice that cost$[s] = c_s$.

Problem 26 (10 points):

The tramp steamer problem. You are the owner of a steamship that can ply between a group of port cities V. You make money at each port: a visit to city i earns you a profit of p_i dollars. Meanwhile, the transportation cost from port i to port j is $c_{ij} > 0$. You want to find a cyclic route in which the ratio of profit to cost is maximized.

To this end, consider a directed graph $G = (V, E)$ whose nodes are ports, and which has edges between each pair of ports. For any cycle C in this graph, the profit-to-cost ratio is

$$r(C) = \frac{\sum_{(i,j)\in C} p_j}{\sum_{(i,j)\in C} c_{ij}}.$$

Let r^* be the maximum ratio achievable by a simple cycle. One way to determine r^* is by binary search: by first guessing some ratio r, and then testing whether it is too large or too small.

Consider any positive $r > 0$. Give each edge (i, j) a weight of $w_{ij} = rc_{ij} - p_j$.

- (a) Show that if there is a cycle of negative weight, then $r < r^*$.
- (b) Show that if all cycles in the graph have strictly positive weight, then $r > r*$.
- (c) Give an efficient algorithm that takes as input a desired accuracy $\epsilon > 0$ and returns a simple cycle C for which $r(C) \geq r^* - \epsilon$. Justify the correctness of your algorithm and analyze its running time in terms of $|V|$, ϵ, and $R = max_{(i,j)\in E}(p_j/c_{ij})$.

Problem 27 (10 points):

Recall that the *CLIQUE* problem is defined as follows: "given a graph $G = (V, E)$ and an integer parameter g, does G contain a clique of size g (i.e., a complete subgraph of G of g vertices)?"

Now 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) Recall that the *VERTEX COVER* problem is defined as follows: "given a graph $G = (V, E)$ and an integer parameter b, does G have a vertex cover of size at most b?"

  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 $O(|V|)$ algorithm for *CLIQUE-3*.

Problem 28 (10 points):

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.

Problem 29 (10 points):

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 the k-SPANNING TREE problem is NP-complete even if we restrict it to the special case $k = 2$.

Problem 30 (10 points):

Show that the following *MAXIMUM COMMON SUBGRAPH* problem is NP-complete.

Input: Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$; an integer parameter 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.

Problem 31 (10 points):

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 an integer parameter g, the *KITE* problem asks for a subgraph which is a kite and which contains $2g$ nodes. Prove that *KITE* is NP-complete.

Problem 32 (10 points):

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. The *FEEDBACK ARC SET (FAS)* problem is defined as follows: "Given a directed graph $G = (V, E)$ and an integer parameter b, find a feedback arc set of $\leq b$ edges, if one exists."

- Task (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 an instance (G', b) of *FAS* as follows. If $G = (V, E)$ has n vertices $v_1, \cdots, v_n$, then make $G' = (V', E')$ a directed graph with $2n$ vertices w_1, w'_1, $\cdots$, w_n, w'_n , and $n + 2|E|$ (directed) edges:

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

Continue with the following tasks:

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

Problem 33 (10 points):

In the *MINIMUM STEINER TREE* problem, the input consists of: a complete graph $G = (V, E)$ with distances d_{uv} between all pairs of nodes; and a distinguished set of terminal nodes $V' \subseteq V$. The goal is to find a minimum-cost tree that includes the vertices V'. This tree may or may not include nodes in $V - V'$.

Suppose the distances in the input are a metric (which means they are symmetric between every pair of nodes and they satisfy the triangle inequality). Show that an efficient ratio-2 approximation algorithm for *MINIMUM STEINER TREE* can be obtained by ignoring the nonterminal nodes and simply returning the minimum spanning tree on V'. (*Hint*: Recall our approximation algorithm for the *TSP*.)

Problem 34 (10 points):

In the *MULTIWAY CUT* problem, the input is an undirected graph $G = (V, E)$ and a set of terminal nodes s_1, s_2, $\cdots$, $s_k \in V$. The goal is to find the minimum set of edges in E whose removal leaves all terminals in different components.

- (a) Show that this problem can be solved exactly in polynomial time when $k = 2$.
- (b) Give an approximation algorithm with ratio at most 2 for the case $k = 3$.

Problem 35 (10 points):

Hollywood. A film producer is seeking actors and investors for his new movie. There are n available actors; actor i charges s_i dollars. For funding, there are m available investors. Investor j will provide p_j dollars, but only on the condition that certain actors $L_j \subseteq \{1, 2, \cdots, n\}$ are included in the cast (all of these actors L_j must be chosen in order to receive funding from investor j). The producer's profit is the sum of the payments from investors minus the payments to actors. The goal is to maximize this profit.

- (a) Express this problem as an integer linear program in which the variables take on values $\{0, 1\}$.
- (b) Now relax this to a linear program, and show that there must in fact be an *integral* optimal solution (as is the case, for example, with maximum flow and bipartite matching).