List of Abbreviations and Symbols

$A[1..n]$	An array indexed from 1 to n of n elements.
$\mathbb{N}$	Set of all natural numbers, i.e., $\{1, 2, 3, \dots\}$.
$\mathbb{R}$	Set of all real numbers.
$\mathbb{Z}$	Set of all integers, i.e., $\{\dots, -2, -1, 0, 1, 2, \dots\}$.
$\exists$	symbol meaning *there exists*.
$\forall$	symbol meaning *for all*.

Modifiers

To help you with what problems to try, problems marked with **[K]** are key questions that tests you on the core concepts, please do them first. Problems marked with **[H]** are harder problems that we recommend you to try after you complete all other questions (or perhaps you prefer a challenge). Good luck!!!

Contents

§1 String matching

Exercise 1.1. [K] Consider the text $T =$ "dbebfjcgfdfj" and pattern $P =$ "dh", taken from the alphabet $\{\text{'a'}, \ldots, \text{'j'}\}$ of size $d = 10$. We will use the Rabin-Karp algorithm to search for matches of P in T, choosing $p = 11$.

By close inspection, you will observe that P does not appear as a substring of T even once (indeed, T does not contain any instances of 'h'). However, the Rabin-Karp algorithm sometimes produces false positives due to hash collisions.

How many false positives are there? That is, letting $T_s = t_s t_{s+1}$ for various starting points s, how many times do we have that $H(T_s) = H(P)$ but $T_s \neq P$?

Exercise 1.2. [K] Let $s = 01011011010$ be a string (of length 11) from the alphabet $\Sigma = \{0, 1\}$.

(a) Compute the transition function for the pattern s, i.e. find $\delta(k, a)$ for all $0 \leq k \leq 11$ and $a \in \Sigma$.

(b) Draw the corresponding state transition diagram. You may omit arrows to state 0.

(c) Compute the prefix function for the pattern s, i.e. find $\pi(k)$ for all $1 \leq k \leq 11$.

Exercise 1.3. [H] Since the Rabin-Karp algorithm functions on a one-dimensional array, explain how you would extend the Rabin-Karp method to look for an $m \times m$ pattern in an $n \times n$ array of symbols.

> **Hint —** *How would you reduce the array to the one-dimensional case?*

Exercise 1.4. [H] Let T and T' be strings of length n. Describe an $O(n)$ time algorithm to determine if T and T' are cyclic rotations of one another. For example, $T =$ car and $T' =$ arc are cyclic rotations of each other, while $T =$ arc and $T' =$ acr are not.

> **Hint —** *If x and y are of the same length, then is it true that x is a cyclic rotation of y precisely when x is a substring of yy?*

Exercise 1.5. [H] Given a list of strings A and a prefix string s, describe an algorithm to count the number of number of strings whose prefix matches s.

Exercise 1.6. [H] Given two patterns T and T', describe how you would construct a finite automaton that determines all occurrences of either T or T'.

Exercise 1.7. [H] If the pattern length is too long or the finite alphabet is relatively large, then Knuth-Morris-Pratt may be somewhat inefficient. Instead of reading the symbols in the forward direction, describe how an algorithm that reads the pattern string in the backward direction may be more efficient.

§2 Linear programming

Exercise 2.1. Sydney Sounds Manufacturing produces speakers for hi-fi sets in two types, product A and product B. Two types of processes are needed for their production. The first process combines all machining operations, and the second consists of all assembly operations. Each unit of product A requires 4 labor-hours of machining and 2 labor-hours of assembly work, whereas each unit of product B requires 3 labor-hours of machining and 0.5 labor-hours of assembly work. Manufacturing capacity available during the coming production week is 2400 labor-hours, and the assembly work capacity available for the same week is 750 labor-hours.

Previous sales experience indicates that product B sells at least as much as product A and that there is already an order of 100 units for product A to be produced during the next period. Furthermore, all products produced during the week can be sold during the same week. Products A and B provide \$7.00 and \$5.00 profit per unit sold, respectively. Management would like to know what quantities of A and B should be manufactured during the next production week in order to maximize the total profit. The following table summarizes the data.

Operation	Product A	Product B	Capacity
Machining (labor-hours)	4	3	2400
Assembly (labor-hours)	2	0.5	750
Profit per unit (\$)	7	5	

(a) **[K]** Write down the linear program P representing this problem in standard form, making sure to define all variables and constraints involved.

(b) **[K]** Write down the dual P^* of the problem P.

(c) **[H]** Find the optimal value of the objective function and the quantities to produce in order to maximize the total profit.

> **Hint —** You can use `MATLAB` or other software to find the solution, however, there is a way to compute it analytically, maybe draw a plot?

Exercise 2.2. [K] You are the boss of a large manufacturing company and you wish to produce a certain amount of items to sell to the customers while making as much profit as possible.

- Based on the technical constraint, you can choose to produce any amount of n different types of items each with a cost of c_i.
- Your accounting department also gives an estimate of the profit for each unit of item i is p_i and your budget is C in total.
- Manufacturing each item i also produces a certain amount of pollution. Based on the constraint set by EPA, there are k many different types of pollution measures and limit you must adhere to for each pollution measure be denoted by L_i.
- Producing each item i will cause any amount of pollution for each of the different measures, we denote this by $(w_{(i,1)}, w_{(i,2)}, \ldots, w_{(i,k)})$.

(a) Classify this problem as either Linear Programming or Integer Linear Programming, and deduce whether there is a known algorithm to solve it in polynomial time?

(b) Formulate this problem P in standard form, making sure to define all variables and constraints involved.

(c) Formulate the dual of this problem P^*.

(d) Suppose that all of a sudden, exactly $m < n$ many types items are required to produce exactly $\nu_1, \nu_2, \ldots \nu_m$ many. How should you modify your LP formulation? (you can assume that producing those items do not exceed the pollution requirement)

Exercise 2.3. Linear programming comes up in financial mathematics, here for this problem we will give a preview of the problem of arbitrage betting. Say there is a huge sports tournament going on with m teams competing and n different betting agencies currently allowing you to place bets on the outcome of the tournament.

(a) **[K]** Suppose that each betting agency i now allows you to put a bet on the jth team wins at the end of the tournament with a payout of $f_{i,j} : 1$, write down a standard form (matrix form) LP problem P that allows you to maximize your risk-free profit with a budget of B.

(b) **[K]** Write down the dual problem P^* for the LP formulation you have derived in **(a)**.

(c) **[H]** Does your LP procedure always give a solution that will make you money? Explain your answer.

> **Hint —** You may want to consider the Arbitrage Theorem.

Exercise 2.4. [H] Linear programming shows up in game theory as well!! Suppose there are n armies advancing on m cities and each army i is commanded by a General G_i with R_i many regiments.

- Each general G_i can send choose to send some amount of regiments to a city.
- In each city, the army that sends the most amount of regiments to the city captures both the city and **all other** army's regiments.
- If there are any ties in the number of regiment for the 1st place, then those corresponding armies draws and loses no points. The rest of the armies are penalised the same amount of points as the regiments sent to that city.
- Each army scores 1 point per city captured and 1 point per captured regiment.

Assume that each army needs to make full use of its regiments, and wants to maximize the sum of the difference between its reward and all other opponent's reward.

(a) This is an example of a zero-sum game. Denote a strategy $\boldsymbol{\pi}_j = (\pi_1, \pi_2, \ldots, \pi_m)$ as follows: the general j sends π_i regiments to city i and so on and let $\mathcal{S}_i$ denote the space of all possible strategies that can be done by general i.

We define the *payoff matrix (or tensor)* $C_k : \mathcal{S}_1 \times \mathcal{S}_2 \times \cdots \times \mathcal{S}_n \to \mathbb{Z}$, which each of the $C(\boldsymbol{\pi}_1, \boldsymbol{\pi}_2, \ldots, \boldsymbol{\pi}_n)$ denotes the payoff of each of the generals selecting the strategy of $\boldsymbol{\pi}_1, \boldsymbol{\pi}_2, \ldots, \boldsymbol{\pi}_n$ in the perspective of the kth general. E.g. If General 1 uses the strategy $(4, 0)$ and General 2 uses the strategy $(3, 0)$ then the corresponding $C((4, 0), (3, 0))$ is -4 as General A captures 1 city and 3 regiments, resulting in a "*loss*" of -4.

Generate the payoff matrix C_1 for $n = 2$, $m = 2$ and $R_1 = 4, R_2 = 3$.

(b) For $n = 2$, write down the standard form LP formulation that finds the optimal strategy for any General in a probabilistic sense.

(c) Outline a method to extend the problem to $n > 2$. You do not need to specify the linear program in full.

§3 Intractability

Exercise 3.1. [K] Determine for each of the following whether it is a polynomial time algorithm.

(a) $O(n)$ input, $O(n \log n)$ running time.

(b) $O(n + \log C)$ input, $O(nC)$ running time.

(c) $O(n)$ input, $O(n^3)$ running time.

(d) $O(n)$ input, $O(2^n)$ running time.

Exercise 3.2. [K] A *clique* of a graph $G = (V, E)$ is a subset $U \subseteq V$ of vertices such that any two elements, $u, v \in U$ are adjacent; that is, for any distinct pairs of vertices $u, v \in U$, there exist an edge such that $(u, v) \in E$ of G.

Consider the optimisation version of the *clique* problem as stated below.

Maximum Clique

Instance: An undirected graph $G = (V, E)$.

Task: Choose a subset $U \subseteq V$ of vertices of maximum size such that for any two vertices $u, v \in U$, we have $(u, v) \in E$.

(a) Convert this optimisation problem to the corresponding decision problem (Clique).

(b) Explain how an algorithm which solves the Clique problem can be extended to solve the original optimisation problem (Maximum Clique) with $\log |V|$ overhead.

(c) Show that the Clique problem is in class **NP**.

Exercise 3.3. [K] Recall the Integer Knapsack problem from the Dynamic Programming lecture. The problem is stated below.

Integer Knapsack

Instance: a positive integer n, integers w_i and v_i for each $1 \leq i \leq n$, and a positive integer C.

Task: Choose a combination of items (with repetition allowed) which all fit in the knapsack and whose total value is as large as possible.

(a) Is this an optimisation or decision problem?

(b) If this is an optimisation problem, convert it to a corresponding decision problem.

(c) Show that the decision problem is in **NP**.

Exercise 3.4. [K] Suppose that U and V are decision problems in class **NP**, and that there exists a polynomial reduction f from U to V. What can you deduce if:

(a) V is in class **P**?

(b) V is in class **NP-C**?

(c) U is in class **P**?

(d) U is in class **NP-C**?

Exercise 3.5. [K] Recall that the Vertex Cover problem is as follows:

Vertex Cover (VC)

Instance: $G = (V, E)$, an undirected and unweighted graph, and a positive integer k.

Task: Is it possible to choose k vertices so that every edge is incident to at least one of the chosen vertices?

In lectures, we showed that the Vertex Cover problem is in **NP**-complete. We will use this problem to prove that a related problem is also **NP**-complete.

Consider the Feedback Vertex Set problem stated below.

Feedback Vertex Set (FVS)

Instance: $G = (V, E)$, an undirected (or directed) graph, and a positive integer k.

Task: Is it possible to choose $k \leq |V|$ vertices so that, if we remove all k vertices and their corresponding adjacent edges from G, the new graph is cycle-free?

(a) Show that Feedback Vertex Set is in **NP**.

(b) We now prove that Feedback Vertex Set is in **NP**-hard. Consider the following reduction.

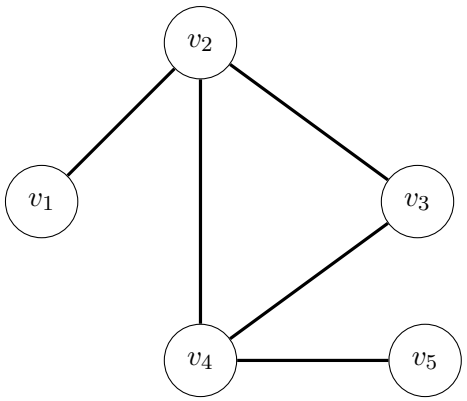
Given a graph $G = (V, E)$, we construct $G' = (V', E')$ where:

- $V' = U_V \cup U_E$, where U_V consists of the vertices of G and, for each edge in $(v_1, v_2) \in E$, we create a new vertex u_{v_1,v_2} which belong in U_E.
- E' consists of the edges constructed as follows:
  - For each edge $(v_1, v_2) \in E$ in the original graph, we construct three edges: an edge from $v_1 \in U_V$ to $v_2 \in U_V$, an edge from $v_1 \in U_V$ to $u_{v_1,v_2} \in U_E$, and an edge from $v_2 \in U_V$ to $u_{v_1,v_2} \in U_E$.

We claim that

$$(G, k) \in \text{VC} \iff (G', k) \in \text{FVS}\,.$$

Consider the following graph G.



(i) Using the reduction described above, construct G'.

(ii) Show that G has a vertex cover of size 2, and find the corresponding feedback vertex set of size 2 in G'. How can we generalise this to arbitrary graphs G, and thus, deduce that Feedback Vertex Set is in **NP**-hard?

(c) Using the previous two results, conclude that Feedback Vertex Set is in **NP**-complete.

Exercise 3.6. [H] Recall that a decision problem X is said to be in **NP** if *yes* instances of X have a certificate and a polynomial-time verifier.

In a similar way, we can then define the following class of problems:

coNP

A decision problem X is said to be in **coNP** if *no* instances of X have a certificate and a polynomial-time verifier.

We say that the *complement* of a problem is the result of switching the "yes" and "no" results. Using the definition of the complement of a problem, we can also define the following class of problems:

coP

A decision problem X is said to be in **coP** if the complement of X is solvable in polynomial time.

(a) Show that $\mathbf{P} = \mathbf{coP}$.

(b) Using part (a), show that, if $\mathbf{P} = \mathbf{NP}$, then $\mathbf{NP} = \mathbf{coNP}$.