COMP3121/9101
Algorithm Design

Problem Set 6 – Selected Topics

[**K**] – key questions [**H**] – harder questions [**E**] – extended questions [**X**] – beyond the scope of this course

Contents

1 SECTION ONE: STRING MATCHING **2**

2 SECTION TWO: LINEAR PROGRAMMING **6**

3 SECTION THREE: INTRACTABILITY **14**

§ SECTION ONE: STRING MATCHING

[K] Exercise 1. Consider the text $T =$ "dbebfjcgfdfj" and pattern $P =$ "dh", taken from the alphabet $\{\text{'a'}, \dots, \text{'j'}\}$ 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 $T_s \neq P$?

Solution. First, convert T and P to the equivalent numbers in base 10, namely 314159265359 and 37.

We observe that $37 \pmod{11} = 4$. So we are looking for substrings of T of length 2 which hash to 4 in modulo 11. We obtain

$$\begin{aligned}
31 &\pmod{11} = 9 \\
14 &\pmod{11} = 3 \\
41 &\pmod{11} = 8 \\
15 &\pmod{11} = 4 \\
59 &\pmod{11} = 4 \\
92 &\pmod{11} = 4 \\
26 &\pmod{11} = 4 \\
65 &\pmod{11} = 10 \\
53 &\pmod{11} = 9 \\
35 &\pmod{11} = 2 \\
59 &\pmod{11} = 4,
\end{aligned}$$

so there are five false positives: "bf", "fj", "jc", "cg" and "fj". □

[K] Exercise 2. 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 $1 \leq k \leq 11$.

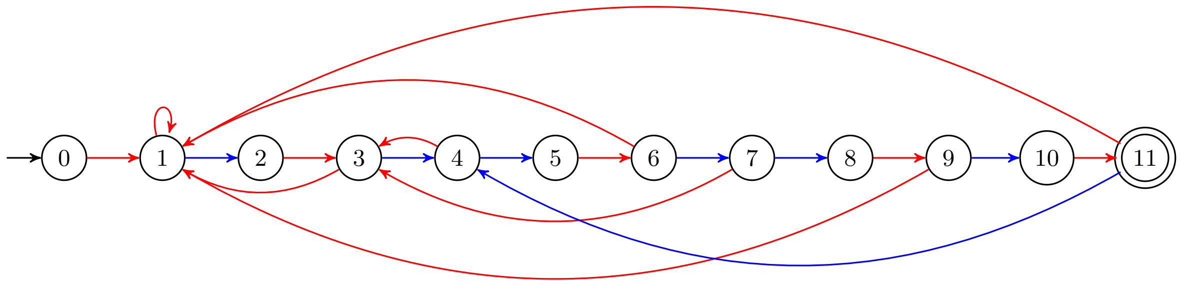
Solution. (a) To compute the transition function, we first compute all of the prefixes of s. We see that the prefixes are

$$\{\epsilon, 0, 01, 010, 0101, 01011, 010110, 0101101, 01011011, 010110110, 0101101101, 01011011010\}.$$

Now, to compute the transition function, we look at the length of the *longest prefix* that can be obtained by appending either a 0 or a 1 at the end of the string that we will have matched. The following transition function is tabulated below.

Length	String matched	**0**	**1**
0	ϵ	1	0
1	0	1	2
2	01	3	0
3	010	1	4
4	0101	3	5
5	01011	6	0
6	010110	1	7
7	0101101	3	8
8	01011011	9	0
9	010110110	1	10
10	0101101101	11	0
11	01011011010	1	4

(b) The diagram is as follows, with red edges representing those labelled 0 and blue edges for those labelled 1.



(c) The prefix function is shown in the following table.

k	1	2	3	4	5	6	7	8	9	10	11
$\pi(k)$	0	0	1	2	0	1	2	0	1	2	3

□

[H] Exercise 3. 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.

Solution. The idea is simple: starting from the first $m \times m$ grid, we move one column along at a time, row by row, because you need to search through all $m \times m$ possible grids in the $n \times n$ grid. For example, let T be the following.

a_{11}	a_{12}	a_{13}
a_{21}	a_{22}	a_{23}
a_{31}	a_{32}	a_{33}

3×3 grid for T

To find a possible 2×2 pattern match, we start with the top left 2×2 grid. We, therefore, search for the following four subgrids.

$$\begin{array}{|c|c|}\hline a_{11} & a_{12} \\ \hline a_{21} & a_{22} \\ \hline\end{array} \Longrightarrow \begin{array}{|c|c|}\hline a_{12} & a_{13} \\ \hline a_{22} & a_{23} \\ \hline\end{array} \Longrightarrow \begin{array}{|c|c|}\hline a_{21} & a_{22} \\ \hline a_{31} & a_{32} \\ \hline\end{array} \Longrightarrow \begin{array}{|c|c|}\hline a_{22} & a_{23} \\ \hline a_{32} & a_{33} \\ \hline\end{array}$$

This intuition will allow us to develop a strategy to extend the Rabin-Karp algorithm. To this end, we convert our $m \times m$ grid into an integer using its m^2 characters. We do this in two parts: the first part, we convert each column into a unique integer; in this way, our m columns give us a unique integer.

In the second stage, we convert our $n \times n$ grid into $m \times m$ blocks. Starting with the top leftmost $m \times m$ block, we convert this into an integer $t_{0,0}$. Then, for each subsequent $m \times m$ block, we can update the value in constant time by shifting the $m \times m$ to the right by one column to obtain $t_{0,1}$. This is done in $O(m)$ time because each column of the $n \times n$ grid correspond to a unique integer which is produced by the m integers. These are used to update subsequent $m \times m$ blocks since $t_{i,j+1}$ can be computed by appealing to the value of $t_{i,j}$.

The above construction is used to compute subsequent $m \times m$ blocks by moving one column to the right. To compute $m \times m$ blocks by moving one row down, we approach it in a similar fashion. Start by updating the m column numbers and compute $t_{i,j}$ for each j. This can be done since each column number is given by the previous row; hence the construction will take $O(m)$ time. The construction can be made unique by using some basis d and the basis numbers d^m are used to compute all $t_{i,j}$ column numbers.

The construction takes polynomial time and we have effectively converted the $n \times n$ case to the one-dimensional Rabin-Karp method. Therefore, to find if the $m \times m$ grid is a subgrid of T, we use a suitable prime modulo q where $q = \Omega(m^2)$. □

[H] Exercise 4. 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.

Solution. If T and T' are cyclic rotations of one another, they must have the same number of characters. Therefore, if $|T| \neq |T'|$, then they can't be cyclic rotations. Now consider the string TT. Our claim is that T and T' are cyclic rotations of one another if and only if T' is a substring of TT.

Indeed, suppose that T and T' are cyclic rotations. Then we can think of T' as shifting T a finite number (say s) of times to the right. In other words, the $|T| - s$ prefix of T must be a suffix of T' and the s suffix of T is a prefix of T'. By concatenating T with itself and by the characterisation above, it follows that T' is a substring of TT.

Suppose that T' is a substring of TT with a shift of s. Then it follows that s suffix of T is a prefix of T', and $|T| - s$ characters left in T' form a prefix of T. In other words, T and T' are cyclic rotations of one another.

Thus, we can use KMP to determine whether T' is a substring in TT, which can be done in linear time. □

[H] Exercise 5. Given a list of strings A and a prefix string s, describe an algorithm to count the number of strings whose prefix matches s.

Solution. Suppose that s is of length k. An intuitive solution is to look for all prefix strings of size k in A. If a string is of smaller length, we discard it from consideration since s cannot be a prefix of such a string. Then, for each prefix string s' in our array, we check for equality. If equality is met, we increment our counter. Otherwise, we continue. The result will count the number of strings with prefix s.

Since each string is truncated to a string of size k, the operation of setting up an auxiliary array (or equivalent) to construct all prefixes of size k is $O(k \cdot |A|)$. Then, for each string in the new array, we check equality in $O(k)$ time. Thus, we also have $O(k \cdot |A|)$ time in the second operation.

Alternatively, we can implement a trie data structure. This will allow us to keep track of *all* possible prefixes. By attaching a counter for each trie node, the counter efficiently counts all strings that have a certain string as its prefix. From this construction, we can find the number of strings that have a certain prefix by traversing from the root to the particular substring in the trie data structure, which has length k. The time complexity is $O(|A| \cdot \ell)$ where ℓ represents the longest word in A. □

[E] Exercise 6. Given two patterns T and T', describe how you would construct a finite automaton that determines all occurrences of either T or T'.

Solution. To give a sketch of the solution, note that we can use KMP to find occurrences of one pattern. Thus, we can employ KMP on T and T' separately. This gives us two automata. We can then build a non-deterministic finite automata that accepts the "union" of the two automata. Starting with the first state (i.e. the state that matches strings of length 0), we non-deterministically choose to match either T or T', and follow along the chosen automaton. Each subsequent state transition is determined uniquely by the transition provided by the corresponding automaton given when running KMP on the respective strings.

The final automaton yields all occurrences of either T or T'. □

§ SECTION TWO: LINEAR PROGRAMMING

[K] Exercise 7. 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) Write down the linear program P representing this problem in standard form, making sure to define all variables and constraints involved.

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

(c) 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?

Solution. (a) Let us define the vector we wish to find as $\mathbf{x} = (x_1, x_2)$ with

- x_1 = number of unit of product A produced per week.
- x_2 = number of unit of product B produced per week.

Basing on the constraints, we formulate the optimization problem as follows

$$\begin{aligned}
\underset{\mathbf{x}\in\mathbb{R}^2}{\text{Maximize}} \quad & f(\mathbf{x}) = 7x_1 + 5x_2 && \text{maximize profit}\\
\text{Subject to} \quad & 4x_1 + 3x_2 \leq 2400, && \text{limit on the capacity for machining}\\
& 2x_1 + \frac{1}{2}x_2 \leq 750 && \text{limit on the capacity for assembly}\\
& x_1 - x_2 \leq 0 && B \text{ sells at least as much as } A\\
& -x_1 \leq -100 && \text{existing orders}\\
& x_1, x_2 \geq 0 && \text{cannot produce negative number of units}
\end{aligned}$$

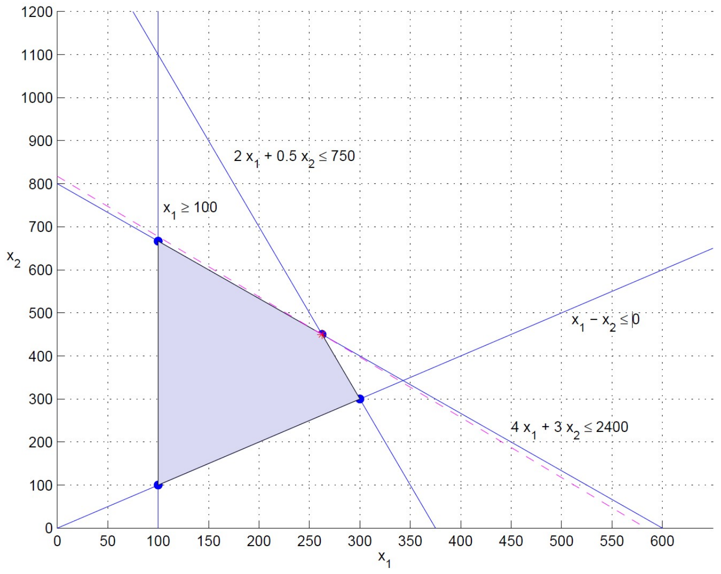
(b) We can write P in the triplet form of $(A, \mathbf{b}, \mathbf{c})$ with

$$\mathbf{c} = \begin{bmatrix} 7 \\ 5 \end{bmatrix} \quad A = \begin{bmatrix} 4 & 3 \\ 2 & \frac{1}{2} \\ 1 & -1 \\ -1 & 0 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} 2400 \\ 750 \\ 0 \\ -100 \end{bmatrix} \quad \mathbf{x} \geq 0.$$

Then, we can easily write its dual problem P^* as

$$\begin{aligned} \underset{\mathbf{y}\in\mathbb{R}^4}{\text{Minimize}} \quad & f^*(\mathbf{y}) = \mathbf{b}^T\mathbf{y} \\ \text{subject to} \quad & A^T\mathbf{y} \geq \mathbf{c} \\ \text{and} \quad & \mathbf{y} \geq 0. \end{aligned}$$

(c) Here we notice that as all the constraints are linear, we can sketch the feasible region of $\mathbf{x}$ as follows:



Note that for P, maximizing $f(\mathbf{x})$ is equivalent to minimizing $-f(\mathbf{x})$. Furthermore, we propose the following:

Claim. For an optimization problem P with a convex function $f(\cdot)$ and convex feasible region Ω, the global minimum must exist on an extreme point of Ω.

Solution. We will omit the proof as it is beyond our scope, but you can read more about it here. □

We can then just simply examine the four points at

$$\mathbf{x} = \begin{pmatrix} 100 \\ 100 \end{pmatrix}, \quad \begin{pmatrix} 300 \\ 300 \end{pmatrix}, \quad \begin{pmatrix} 262.5 \\ 450 \end{pmatrix}, \quad \begin{pmatrix} 100 \\ \frac{2000}{3} \end{pmatrix}.$$

Then in this case our optimal solution yields

$$\mathbf{x}^* = \begin{pmatrix} 262.5 \\ 450 \end{pmatrix} \qquad f(\mathbf{x}^*) = 4087.5.$$

□

[K] Exercise 8. Suppose that there are 3 farmers each with a square of land with the side length of s_i and center of the land at (x_i, y_i) However, due to the current drought, the farmers need to drill a single well to extract water and keep the crops alive. The well's position must be chosen such that it is within the land of all 3 farmers. The amount of water that is extractable from a coordinate (x, y) can be written as a $w = 4x + 6y$ and you may assume that no land will include negative coordinates.

(a) Is there always a valid solution for the position of the well? Explain your answer.

(b) Suppose a valid solution exists, derive the standard form LP formulation P that finds the correct position to drill the well such that most amount of water can be extracted while satisfying the requirement.

(c) How can you extend the definition of this problem to n farmers?

Solution. (a) There exists no solution if there exists there are no overlap between the lands of any pair of farmers. Formally, let $r_i = s_i/2$, if exists farmer i and farmer j such that for

$$\begin{aligned} O_{\text{row}} &= (x_i - r_i \leq x_j + r_j) \wedge (x_j - r_j \leq x_i + r_i) \\ O_{\text{col}} &= (y_i - r_i \leq y_j + r_j) \wedge (y_j - r_j \leq y_i + r_i) \\ O^* &= O_{\text{row}} \wedge O_{\text{col}}. \end{aligned}$$

we do not have $O^* = T$, then there is no suitable position for the well.

(b) We start by noting that our constraint requires that

$$|x - x_i| \leq r_i \quad |y - y_i| \leq r_i \quad \text{for } i = 1, 2, 3$$

hence (x, y) must be within the field of each farmer. However this is not in standard form, we need to transform each of the clauses to

$$|x - x_i| \leq r_i \implies x - x_i \leq r_i \quad \text{and} - x + x_i \leq r_i$$

Therefore, we can write the standard form LP P for $\mathbf{x} = (x, y)$,

$$\begin{aligned} \underset{\mathbf{x}\in\mathbb{R}^2}{\text{Maximize}} \quad & f(\mathbf{x}) = 4x + 6y & & \text{maximize } w \text{ the amount of water} \\ \text{Subject to} \quad & x - x_i \leq r_i \quad \text{for } i = 1, 2, 3, \\ & y - y_i \leq r_i \quad \text{for } i = 1, 2, 3. \\ & -x + x_i \leq r_i \quad \text{for } i = 1, 2, 3, \\ & -y + y_i \leq r_i \quad \text{for } i = 1, 2, 3. \\ & x, y \geq 0 & & \text{no negative coordinates.} \end{aligned}$$

We can then write down the triplet form of $(\mathbf{c}, A, \mathbf{b})$ with

$$\mathbf{c} = \begin{pmatrix} 4 \\ 6 \end{pmatrix} \quad A = \begin{pmatrix} I_{2\times2} \\ I_{2\times2} \\ I_{2\times2} \\ -I_{2\times2} \\ -I_{2\times2} \\ -I_{2\times2} \end{pmatrix} \quad \mathbf{b} = \begin{pmatrix} r_1 + x_1 \\ r_1 + y_1 \\ \vdots \\ r_3 + x_3 \\ r_3 + y_3 \\ r_1 - x_1 \\ r_1 - y_1 \\ \vdots \\ r_3 - x_3 \\ r_3 - y_3 \end{pmatrix}.$$

With $I_{k\times k}$ as the identity matrix of size k.

Hint. Is there a way to compute the feasible region directly?

(c) Natural extension to n farmers will require

- Our condition are satisfied for (a).

- Our LP problem then becomes

$$\begin{aligned} \underset{\mathbf{x}\in\mathbb{R}^2}{\text{Maximize}} \quad & f(\mathbf{x}) = 4x + 6y \\ \text{Subject to} \quad & x - x_i \leq r_i \quad \text{for } i = 1, \dots n, \\ & y - y_i \leq r_i \quad \text{for } i = 1, \dots, n. \\ & -x + x_i \leq r_i \quad \text{for } i = 1, \dots, n, \\ & -y + y_i \leq r_i \quad \text{for } i = 1, \dots, n. \\ & x, y \geq 0 \end{aligned}$$

Our matrix also changes to

$$A = (\underbrace{I_{2\times2}, \dots, I_{2\times2}}_{n \text{ times}}, \underbrace{-I_{2\times2}, \dots, -I_{2\times2}}_{n \text{ times}})^T \quad \mathbf{b} = \begin{pmatrix} r_1 + x_1 \\ r_1 + y_1 \\ \vdots \\ r_n + x_n \\ r_n + y_n \\ r_1 - x_1 \\ r_1 - y_1 \\ \vdots \\ r_n - x_n \\ r_n - y_n \end{pmatrix}.$$

as required.

□

[K] Exercise 9. 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 $(w_{(i,1)}, w_{(i,2)}, \dots, w_{(i,k)})$.

(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 $\nu_1, \nu_2, \dots \nu_m$ many. How should you modify your LP formulation? (you can assume that producing those items do not exceed the pollution requirement)

Solution. (a) Since we cannot produce half an item, this problem is an **Integer Linear programming** problem, hence there is no known polynomial time algorithm for solving it.

(b) We start by defining our variables, let x_i denote the number of units of item type i. Then with $\mathbf{x} = (x_1, \dots, x_n)$ We can formulate our standard form LP P as

$$\begin{aligned}
\underset{\mathbf{x}\in\mathbb{Z}^n}{\text{Maximize}} \quad & f(\mathbf{x}) = \sum_{i=1}^{n} x_i p_i && \text{maximize the profit obtainable}\\
\text{Subject to} \quad & \sum_{i=1}^{n} c_i x_i \le C && \text{no spending exceeds } C\\
& \sum_{i=1}^{n} w_{(i,j)} x_i \le L_j \quad \text{for } j = 1, \dots, k && \text{cannot exceed total pollution limit}\\
& \mathbf{x} \ge 0 && \text{cannot manufacture negative units}
\end{aligned}$$

Then we can obtain its matrix triplet,$(\mathbf{c}, A, \mathbf{b})$ namely

$$\mathbf{c} = \begin{pmatrix} p_1 \\ \vdots \\ p_n \end{pmatrix} \quad A = \begin{pmatrix} c_1 & c_2 & c_3 & \dots & c_n \\ w_{(1,1)} & w_{(2,1)} & w_{(3,1)} & \cdots & w_{(n,1)} \\ w_{(1,2)} & w_{(2,2)} & w_{(3,2)} & \cdots & w_{(n,2)} \\ w_{(1,3)} & w_{(2,3)} & w_{(3,3)} & \cdots & w_{(n,3)} \\ \vdots & \dots & \dots & \ddots & \vdots \\ w_{(1,k)} & w_{(2,k)} & w_{(3,k)} & \cdots & w_{(n,k)} \end{pmatrix} = \begin{pmatrix} \gamma_{1,n} \\ W_{k,n} \end{pmatrix} \quad \mathbf{b} = \begin{pmatrix} C \\ L_1 \\ L_2 \\ L_3 \\ \vdots \\ L_k \end{pmatrix}$$

(c) From the matrix triplet we have derived above, the dual of P is then

$$\begin{aligned}
\underset{\mathbf{y}\in\mathbb{Z}^{k+1}}{\text{Minimize}} \quad & f(\mathbf{x}) = \mathbf{b}^T \mathbf{y}\\
\text{Subject to} \quad & A^T \mathbf{y} \ge \mathbf{c}\\
& \mathbf{y} \ge 0
\end{aligned}$$

(d) Here we modify our P such that it adheres to the equality constraint. Without loss of generality, let $1, \dots, m$ denote the items types that requires exact units of production and let $m+1, \cdots, n$ be the unrestricted types. We can then reduce our optimization problem P' with $\mathbf{z} = (z_1, \dots, z_{n-m})$

$$\begin{aligned}
\underset{\mathbf{z}\in\mathbb{Z}^{n-m}}{\text{Maximize}} \quad & f(\mathbf{x}) = \sum_{i=1}^{n-m} p_{m+i} z_i\\
\text{Subject to} \quad & \sum_{i=1}^{n-m} z_i c_{m+i} \le C - \sum_{i=1}^{m} c_i \nu_i,\\
& \sum_{i=1}^{n-m} w_{(m+i,j)} z_i \le L_j - \sum_{i=1}^{m} w_{(i,j)} \nu_i \quad \text{for } j = 1, \dots, k,\\
& \mathbf{z} \ge 0.
\end{aligned}$$

Or equivalently, we can also modify our problem by adding $2m$ many restrictions such that $\mathbf{x}_i \le \nu_i$ and $\mathbf{x}_i \ge \nu_i$ for $i = 1, \dots, m$ to enforce the equality constraint. The matrix form of P' will be *left as an exercise for the reader.*

□

[H] Exercise 10. 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) Suppose that each betting agency i now allows you to put a bet on the t_ith team wins at the end of the tournament with a payout of $f_i : 1$, write down a standard form (matrix form) LP problem P that allows you to maximize your risk-free profit (i.e., a strategy that yields a profit regardless of the outcomes) with a budget of B.

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

(c) 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.

Solution. (a) We start by defining our variables, let $\mathbf{x} = (x_1, x_2, \dots, x_n)$ be the amount placed at the nth betting agency. We then define another set of variables r which

$$r_{(i,j)} = \begin{cases} f_i - 1 & \text{if } j = t_i \\ -1 \end{cases}$$

which is the return for a bet of \$1 on agency i if the outcome is j. We now define the concept of a composite return for betting on a specific outcome j as

$$\varphi_j = \sum_{i=1}^{n} x_i r_{(i,j)}.$$

therefore, a risk free strategy will require that $\varphi_j > 0$ for all $j = 1, \dots, m$. However, we will also introduce x_{n+1} as the minimal profit for each outcome. We can then formulate our LP problem P,

$$\begin{array}{lll} \underset{\mathbf{x}\in\mathbb{R}^{n+1}}{\text{Maximize}} & f(\mathbf{x}) = x_{n+1} & \text{maximize minimal profit} \\ \text{Subject to} & \varphi_j \geq x_{n+1} \quad \text{for } j = 1, \dots, m & \text{optimize the profit for each outcome} \\ & \sum_{i=1}^{n} x_i \leq B & \text{cannot spend over budget} \\ & \mathbf{x} \geq 0 & \text{cannot give betting agency money} \end{array}$$

Or equivalently, we can write this problem in the triplet form via

$$\mathbf{c} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{pmatrix} \quad A = \begin{pmatrix} -r_{(1,1)} & -r_{(2,1)} & \cdots & -r_{(n,1)} & 1 \\ -r_{(1,2)} & -r_{(2,2)} & \cdots & -r_{(n,2)} & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ -r_{(1,m)} & -r_{(2,m)} & \cdots & -r_{(n,m)} & 1 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix} \quad \mathbf{b} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ B \end{pmatrix}.$$

(b) The dual P^* can again be derived directly from the matrix form via

$$\begin{array}{ll} \underset{\mathbf{y}\in\mathbb{R}^{m+1}}{\text{Minimize}} & f(\mathbf{x}) = \mathbf{b}^T\mathbf{y} \\ \text{Subject to} & A^T\mathbf{y} \geq \mathbf{c} \\ & \mathbf{y} \leq 0 \end{array}$$

(c) We can check the conditions outlined by a simplified version of the Arbitrage Theorem.

Theorem (Arbitrage Theorem). For the discrete probability $p_1, p_2, \ldots, p_m$ for the outcome of the tournament, we have that either

(1) there is a betting strategy $\mathbf{x}$ which

$$\sum_{i=1}^{n} x_i r_{(i,j)} > 0 \quad \text{for } j = 1, \ldots, m$$

(2) or we have that

$$\sum_{j=1}^{m} p_j r_{(i,j)} = 0 \quad \text{for } i = 1, \ldots, n$$

So (2) clause tells us that no matter how we bet, our expected return is always 0.

$\square$

[H] Exercise 11. 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 $\pi_j = (\pi_1, \pi_2, \ldots, \pi_m)$ 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 $C(\pi_1, \pi_2, \ldots, \pi_n)$ denotes the payoff of each of the generals selecting the strategy of $\pi_1, \pi_2, \ldots, \pi_n$ in the perspective of G_k. E.g. If General 1 uses the strategy $(4, 0)$ and General 2 uses the strategy $(3, 0)$ then the corresponding $C_2((4, 0), (3, 0))$ is -4 as General A captures 1 city and 3 regiments, resulting in a *loss* of -4.

Generate the payoff matrix C_2 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.

Solution. (a) We start by computing all possible strategies of both generals

- G_1's strategies are: $(4, 0), (0, 4), (3, 1), (1, 3), (2, 2)$
- G_2's strategies are: $(3, 0), (0, 3), (2, 1), (1, 2)$

Then the payoff matrix is:

	(3,0)	(0,3)	(2, 1)	(1, 2)
(4, 0)	-4	0	-2	-1
(0, 4)	0	-4	-1	-2
(3, 1)	-1	1	-3	0
(1, 3)	1	-1	0	-3
(2, 2)	2	2	-2	-2

(b) We start by defining our variables,

- let $\mathbf{y} = (y_1, \dots, y_M)^T \in \mathbb{R}^5$, with y_i be the probability that G_1 chooses the strategy specified by row i
- let $\mathbf{x} = (x_1, \dots, x_N)^T \in \mathbb{R}^4$, with x_i be the probability that G_2 chooses the strategy specified by column i

Let P be the payoff matrix we generated similarly to (a), we can solve this problem via the **Weak Duality Theorem**!! Let us define a standard form of $P(f, \mathbf{y})$ as

$$\begin{aligned}
&\underset{\mathbf{x}\in\mathbb{R}^N}{\text{Maximize}} && f(\mathbf{x}) = \mathbf{y}^T P\mathbf{x} && \text{maximize score in perspective of } G_2\\
&\text{Subject to} && \sum_{i=1}^{N} x_i = 1 && \mathbf{x} \text{ is a probability distribution}\\
& && \mathbf{x} \geq 0 && \text{probability is positive}
\end{aligned}$$

Let us define a standard form of $Q(f, \mathbf{x})$ as

$$\begin{aligned}
&\underset{\mathbf{y}\in\mathbb{R}^M}{\text{Minimize}} && f(\mathbf{y}) = \mathbf{y}^T P\mathbf{x} && \text{minimize the score of } G_1\\
&\text{Subject to} && \sum_{i=1}^{M} y_i = 1 && \mathbf{y} \text{ is a probability distribution}\\
& && \mathbf{y} \geq 0 && \text{probability is positive}
\end{aligned}$$

We can then solve this problem directly via Weak duality between P and Q. However, we can actually simplify this further, note that $\mathbf{y}$ must be a proper discrete probability distribution, we can expand f and obtain the LP form as below P^*

$$\begin{aligned}
&\underset{\mathbf{x}\in\mathbb{R}^N, r\in\mathbb{R}}{\text{Maximize}} && f(\mathbf{x}) = r\\
&\text{Subject to} && (P\mathbf{x})^T \mathbf{e}_i \geq r \quad \text{for } i = 1, \dots, N\\
& && \sum_{j=1}^{N} x_j = 1\\
& && \mathbf{x} \geq 0
\end{aligned}$$

With $\mathbf{e}_i$ as the ith vector in standard basis in $\mathbb{R}^N$. For optimal solution for G_1 we can replicate the above process but in reverse.

□

§ SECTION THREE: INTRACTABILITY

[K] Exercise 12. 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.

Solution. (a) Yes, the running time is bounded by n^2.

(b) No, the running time is not bounded by any polynomial in n and $\log C$, since C is exponential in $\log C$.

(c) Yes, the running time is bounded by n^3.

(d) No, the running time is not bounded by any polynomial in n.

□

[K] Exercise 13. 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**.

Solution. (a) The decision problem is

Clique

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

Task: Choose a subset $U \subseteq V$ consisting of at least k vertices (if one exists) such that for any two vertices $u, v \in U$, we have $(u, v) \in E$.

(b) Binary search for the largest k for which a clique of size $\geq k$ exists.

(c) The certificate is a claimed clique U. The certifier problem is

Clique Certifier

Instance: An undirected graph $G = (V, E)$, an integer k, and a subset $U \subseteq V$ of the vertices.

Task: Determine whether U contains at least k vertices and whether for any two vertices $u, v \in U$ we have $(u, v) \in E$.

This is solved in $O(n^2)$ by first making an adjacency matrix to represent G, then counting the vertices of U and finally checking whether an edge appears between each pair of vertices of U. The runtime is clearly polynomial in the length of the input ($O(n + m)$).

□

[K] Exercise 14. 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 $1 \le i \le n$, 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**.

Solution. (a) This is an optimisation problem, since we are maximising the total value.

(b) The corresponding decision problem is as follows.

Integer Knapsack (Decision)

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

Task: Determine whether there is a combination of items (with repetition allowed) which all fit in the knapsack and whose total value is at least V.

(c) Consider the certificate $\mathbf{a} = (a_1, a_2, \dots, a_n)$, corresponding to picking the ith item a_i many times. The certifier problem is then to check whether such a collection has total weight

$$\sum_{i=1}^{n} a_i w_i \le C$$

and total value

$$\sum_{i=1}^{n} a_i v_i \ge V,$$

both of which can be determined in $O(n)$. Thus Integer Knapsack (Decision) is in **NP**.

□

[K] Exercise 15. 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**?

Solution. (a) If V is in the class **P**, there is a polynomial time algorithm to solve instances of V. Now, for an instance x of problem U, first find the corresponding instance $f(x)$ of problem V, then solve it. This algorithm is also polynomial time, so U is in class **P**.

(b) If V is in class **NP-C**, then by definition any problem in **NP** is polynomially reducible to V. Therefore, we cannot deduce anything further about U.

(c) We cannot deduce anything further about V from the assumption that U is in class **P**.

(d) If U is in class **NP-C**, then for any **NP** problem W there is a polynomial reduction g from W to U. By composition, we obtain the polynomial reduction $f \circ g$ from W to V, so V is in **NP-C** also.

□

[H] Exercise 16. 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.

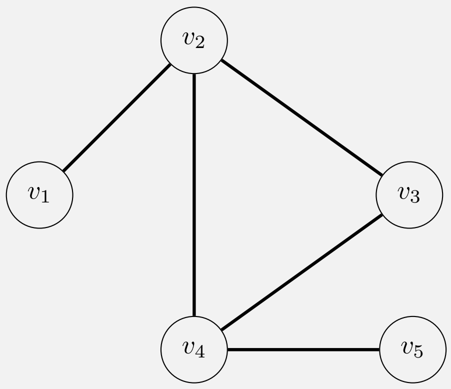
Given a graph $G = (V, E)$, we construct $G' = (V', E')$ where:

- $V' = U_V \cup U_E$, 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.



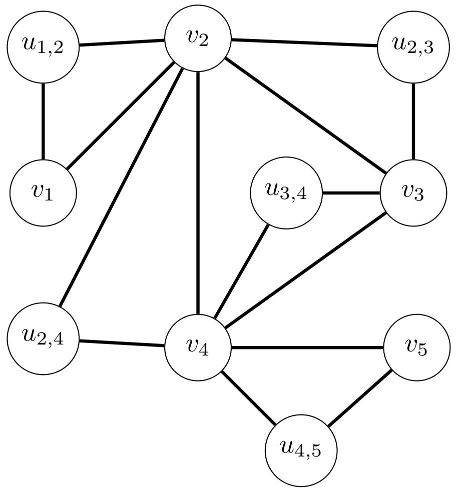
(i) Using the reduction described above, construct G'.

(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.

Solution. (a) To show that Feedback Vertex Set is in **NP**, we construct a certificate and a polynomial time verifier. The certificate is the subset of k vertices. To verify whether this is a feedback vertex set, we remove these vertices from G to construct G'. Then run a depth-first search on G' to determine if there are cycles. Since G is a graph with a finite number of vertices and edges, so is G'; the depth-first search takes polynomial time on G'. Thus, Feedback Vertex Set is in **NP**.

(b) (i) We construct the following graph, G'.



(ii) In G, we note that the set $S = \{v_2, v_4\}$ is a vertex cover of size 2 because every edge is incident to some vertex in S. This is easy to check. In G', note that the cycles we construct are caused by edges that were present in the original graph. This tells us that, if we had a vertex cover set in G, then the corresponding vertices in S also correspond to the vertices of the Feedback Vertex Set of G'. In other words, the corresponding feedback vertex set is $S' = \{v_2, v_4\}$.

To see how we generalise this result to arbitrary graphs, we prove that the reduction is valid. In this course, we give a high level reasoning for why $(G, k) \in \text{VC} \iff (G', k) \in \text{FVS}$.

The reduction works by forming small 3-cycles, $\{v_i, u_{i,j}, v_j\}$, that correspond *precisely* to the edge $(v_i, v_j) \in E$ in the original graph, G. If we had a vertex cover S in G, then such an edge must be covered by either v_i or v_j belonging to S. Thus, to break the cycle, we need to choose at least one of v_i, v_j, or $u_{i,j}$. By choosing the vertex belonging to our vertex cover, we ensure that the 3-cycle will be broken if we remove such a vertex in G'. In other words, breaking cycles of G' is equivalent to choosing the vertices that cover every edge of G.

This shows that the Feedback Vertex Set is as hard as solving Vertex Cover. In other words, Feedback Vertex Set is in **NP**-hard.

(c) Since Feedback Vertex Set is an **NP** problem (from part (a)) and can be reduced from an **NP**-complete problem (from part (b)), Feedback Vertex Set is consequently in **NP**-complete.

□

[E] Exercise 17. 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 $\mathbf{NP} = \mathbf{coNP}$.

Solution. (a) We first show that $\mathbf{P} \subseteq \mathbf{coP}$. Consider any problem $X \in \mathbf{P}$. Then X can be solved in polynomial time. In other words, there exist a polynomial running time algorithm that solves (or decides) "yes" and "no" instances of X. However, just by switching the "yes" and "no" instances of the algorithm gives you a polynomial time algorithm that solves the complement of X. In other words, if $X \in \mathbf{P}$, then $\bar{X} \in \mathbf{P}$ or equivalently, $X \in \mathbf{coP}$. This shows that $\mathbf{P} \subseteq \mathbf{coP}$.

The reverse argument shows that $\mathbf{coP} \subseteq \mathbf{P}$. This shows that $\mathbf{P} = \mathbf{coP}$. The formal argument requires an introduction to Turing machines; however, this is a fairly informal discussion on why $\mathbf{P} = \mathbf{coP}$.

(b) Assuming that $\mathbf{P} = \mathbf{NP}$, we have the following

$$\begin{aligned} \mathbf{NP} &= \mathbf{P} && \text{(by assumption)} \\ &= \mathbf{coP} && \text{(from part (a))} \\ &= \mathbf{coNP}. && \text{(since } \mathbf{P} = \mathbf{NP}\text{)} \end{aligned}$$

□