1 Study Group

List the names and SIDs of the members in your study group. If you have no collaborators, you must explicitly write “none”.

2 Egg Drop Revisited

Recall the Egg Drop problem from Homework 6:

You are given \( k \) identical eggs and an \( n \) story building. You need to figure out the highest floor \( \ell \in \{0, 1, 2, \ldots, n\} \) that you can drop an egg from without breaking it. Each egg will never break when dropped from floor \( \ell \) or lower, and always breaks if dropped from floor \( \ell + 1 \) or higher. \( \ell = 0 \) means the egg always breaks. Once an egg breaks, you cannot use it any more. However, if an egg does not break, you can reuse it.

Let \( f(n, k) \) be the minimum number of egg drops that are needed to find \( \ell \) (regardless of the value of \( \ell \)).

Instead of solving for \( f(n, k) \) directly, we define a new subproblem \( M(x, k) \) to be the maximum number of floors for which we can always find \( \ell \) in at most \( x \) drops using \( k \) eggs.

(a) Find a recurrence relation for \( M(x, k) \) that can be computed in constant time given the previous subproblems. Briefly justify your recurrence.

\( \text{Hint: As a starting point, what is the highest floor that we can drop the first egg from and still be guaranteed to solve the problem with the remaining } x - 1 \text{ drops and } k - 1 \text{ eggs if the egg breaks?} \)

(b) Give an algorithm to compute \( M(x, k) \) given \( x \) and \( k \) and analyze its runtime.

(c) Modify your algorithm from (b) to compute \( f(n, k) \) given \( n \) and \( k \).

\( \text{Hint: If we can find } \ell \text{ when there are more than } n \text{ floors, we can also find } \ell \text{ when there are } n \text{ floors.} \)

(d) Analyze the runtime of computing \( f(n, k) \) using the algorithm from last week’s Egg Drop problem.

(e) Show that the runtime of the algorithm of part (c) is \( O(nk) \). How can we implement the algorithm using \( O(k) \) space? Also, compare this runtime of \( O(nk) \) to the runtime that you found in part (d).
3 Knightmare

Give a dynamic programming algorithm to find the number of ways you can place knights on an $M$ by $L$ ($L < M$) chessboard such that no two knights can attack each other (there can be any number of knights on the board, including zero knights). Knights can move in a $2 \times 1$ shape pattern in any direction. Clearly describe your algorithm, prove its correctness, and analyze its runtime as well as space complexity. Your algorithm’s runtime can be exponential in $L$ but should be polynomial in $M$. Return your answer mod 1337.

For this problem, write your answer in the following 4-part format:

(a) Define a function $f(\cdot)$ in words, including how many parameters are and what they mean, and tell us what inputs you feed into $f$ to get the answer to your problem.

(b) Write the “base cases” along with a recurrence relation for $f$.

(c) Prove that the recurrence correctly solves the problem.

(d) Analyze the runtime and space complexity of your final DP algorithm. Can the bottom-up approach to DP improve the space complexity? (For example, we saw in class that Knapsack can be solved in $O(nW)$ space, but one can reduce that to $O(W)$ space via the optimization of only including the last two rows.)

Hint: if a knight is on row $i$, what rows on the chessboard can it affect?
4 Standard Form LP

Recall that any Linear Program can be reduced to a more constrained standard form where all variables are nonnegative, the constraints are given by equations and the objective is that of minimizing a cost function.

More formally, our variables are $x_i$. Our objective is $\min \; c^T x = \sum_i c_i x_i$ for some constants $c_i$. The $j$th constraint is $\sum_i a_{ij} x_i = b_j$ for some constants $a_{ij}, b_j$. Finally, we also have the constraints $x_i \geq 0$.

An example standard form LP:

$$\text{minimize } 5x_1 + 3x_2$$
$$\text{subject to } \begin{cases} x_1 + x_2 - x_3 = 1 \\ -(x_1 + x_2 - x_3) = -1 \\ -x_1 + 2x_2 + x_4 = 0 \\ -(x_1 + 2x_2 + x_4) = 0 \\ x_1, x_2, x_3, x_4 \geq 0 \end{cases}$$

For each of the subparts, what system of variables, constraints, and objectives would be equivalent to the following:

(a) Max Objective: $\max \sum_i c_i x_i$

(b) Upper Bound on Variable: $x_1 \leq b_1$

(c) Lower Bound on Variable: $x_2 \geq b_2$

(d) Bounded Variable: $b_2 \leq x_3 \leq b_1$

(e) Inequality Constraint: $x_1 + x_2 + x_3 \leq b_3$

(f) Min Max Objective: $\min \max (y_1, y_2)$

(g) Unbounded Variable: $x_4 \in \mathbb{R}$
5 Baker

You are a baker who sells batches of brownies and cookies (unfortunately no brookies... for now). Each brownie batch takes 4 kilograms of chocolate and 2 eggs to make; each cookie batch takes 1 kilogram of chocolate and 3 eggs to make. You have 80 kilograms of chocolate and 90 eggs. You make a profit of 60 dollars per brownie batch you sell and 30 dollars per cookie batch you sell, and want to figure out how many batches of brownies and cookies to produce to maximize your profits.

(a) Formulate this problem as a linear programming problem. Draw the feasible region, and find the solution (state the cost function, linear constraints, and all vertices except for the origin).

(b) Suppose instead that the profit per brownie batch is $C$ dollars and the profit per cookie batch remains at 30 dollars. For each vertex you listed in the previous part, give the range of $C$ values for which that vertex is the optimal solution.
6 Meal Replacement

Jonny is planning an "Introduction to CS Theory" overnight summer camp for penguins in Antarctica. Hungry penguins can’t solve problems well, so Jonny is securing an emergency source of food in case polar bears sneak in and eat everything in the igloo. Unfortunately, he is on a tight budget, and in order to accommodate as many penguins as possible, he needs to minimize the cost of food while still meeting the penguins’ minimum dietary needs.

Every penguin needs to consume at least 500 calories of protein per day, 800 calories of carbs per day, and 700 calories of fat per day. Jonny has three options for food he’s considering buying: salmon, bread, and squid. The composition of each food is provided in the following table:

<table>
<thead>
<tr>
<th>Food Type</th>
<th>Price</th>
<th>Protein Calories per Pound</th>
<th>Carb Calories per Pound</th>
<th>Fat Calories per Pound</th>
</tr>
</thead>
<tbody>
<tr>
<td>Salmon</td>
<td>5</td>
<td>500</td>
<td>0</td>
<td>500</td>
</tr>
<tr>
<td>Bread</td>
<td>2</td>
<td>50</td>
<td>300</td>
<td>25</td>
</tr>
<tr>
<td>Squid</td>
<td>4</td>
<td>300</td>
<td>100</td>
<td>200</td>
</tr>
</tbody>
</table>

Our goal is to find a combination of these options that meets the penguins’ daily dietary needs while being as cheap as possible.

(a) Formulate this problem as a linear program.

(b) Take the dual of your LP from part (a).

(c) Suppose now there is a pharmacist trying to assign a price to three pills, with the hopes of getting us to buy these pills instead of food. Each pill provides exactly one of protein, carbs, and fiber.

Interpret the dual LP variables, objective, and constraints as an optimization problem from the pharmacist’s perspective.
7 Coding Questions

For this week’s coding questions, we’ll be implementing **Edit Distance** and **Knapsack**. We will be working with the `hw7_coding.ipynb` notebook in the `hw7` folder [here]. You can also download the file to your personal computer from [here] and complete the exercises locally. Please complete the Jupyter Notebook and make sure to submit your work for each problem to the corresponding subpart on Gradescope.

(a) **Edit Distance**

(b) **Knapsack**

Notes:

- **Submission Instructions**: Please merge your completed Jupyter Notebook with your written solutions for other questions and submit one merged pdf file to Gradescope.

- **OH/HWP Instructions**: While we will be providing conceptual help on the coding portion of homeworks, OH staff will not look at your code and/or help you debug.

- **Academic Honesty Guideline**: We realize that code for some of the algorithms we ask you to implement may be readily available online, but we strongly encourage you to not directly copy code from these sources. Instead, try to refer to the resources mentioned in the notebook and come up with code yourself. That being said, we do acknowledge that there may not be many different ways to code up particular algorithms and that your solution may be similar to other solutions available online.