CS 170 Homework 8

Due **3/21/2023, at 10:00 pm (grace period until 11:59pm). This homework is hard, so we highly recommend starting early.**

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 Discussion 7:

> *You are given k identical eggs and an n story building. You need to figure out the highest floor $\ell \in \{0, 1, 2, \dots n\}$ that you can drop an egg from without breaking it. Each egg will never break when dropped from floor ℓ 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 ℓ (regardless of the value of ℓ).*

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 ℓ in at most x drops using k eggs. In other words, you can think of $M(x, k)$ as representing the maximum number of floors "covered" by using x drops with 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.

Hint: As a starting point, suppose we drop our first egg somewhere and it breaks. How many drops and eggs do you have left? What is the maximum number of floors covered by these remaining drops and eggs?

(b) If we want to use dynamic programming to compute $M(x, k)$ given x and k, in what order do we solve the subproblems? What is the runtime of this DP algorithm?

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

Hint: If we can find ℓ when there are more than n floors, we can also find ℓ when there are n floors.

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

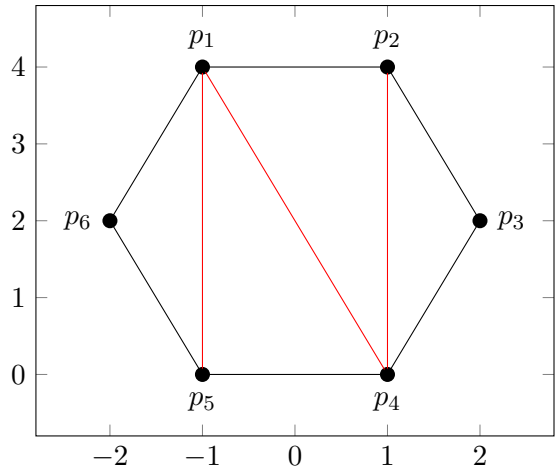
(e) Analyze the runtime of computing $f(n, k)$ using the algorithm from part (c). Also, describe how we can implement this algorithm to use only $O(k)$ space.

(f) Compare the runtimes that you found in parts (d) and (e).

3 Triangulating a Polygon

You are given a convex polygon P that has n vertices $p_1 = (x_1, y_1), \ldots, p_n = (x_n, y_n)$.. We define a triangulation of P to be a collection T of $n-3$ diagonals from P such that no two diagonals intersect inside the polygon. A triangulation splits the polygon's interior into $n-2$ disjoint triangles.

For example, suppose we have a hexagon $P = [p_1 = (-1, 4), p_2 = (1, 4), p_3 = (2, 2), p_4 = (1, 0), p_5 = (-1, 0), p_6 = (-2, 2)]$. One possible triangulation T is $\{(p_1, p_4), (p_1, p_5), (p_2, p_4)\}$, denoted in red in the depiction below:



Now, we define the cost of a triangulation to be the sum of the lengths of the diagonals forming the triangulation:

$$\text{cost}(T) := \sum_{(p_i, p_j) \in T} \ell(p_i, p_j)$$

where the length of a diagonal connecting $p_i = (x_i, y_i)$ and $p_j = (x_j, y_j)$ is $\ell(p_i, p_j) = \sqrt{|x_i - x_j|^2 + |y_i - y_j|^2}$, the Euclidean distance between the two points.

Your task is to devise a dynamic programming algorithm to compute the minimum cost of any triangulation of a given polygon P, i.e. $\min_T\{\text{cost}(T)\}$. **Please provide a three part solution.**

Hint: First order the points $(x_1, y_1), \ldots, (x_n, y_n)$ in a clock-wise manner. Then, you want to index each of your subproblems with a pair of indices $1 \le i < j \le n$. If you are stuck, reading about Chain Matrix Multiplication in DPV section 6.5 may help guide your thinking.

4 A Coin Game

Consider the following single-player game. There are two fair coins A and B. The two sides of A are numbered with 1 and 2, while those of B are numbered with -1 and -2. In each round of the game, you can choose to flip only coin A, only coin B, both coins, or neither. The numbers you get from flipping the coins are added to a running total.

Given a nonnegative integer R and another integer $N \in [-2R, 2R]$, our goal is to design a strategy for the game that maximizes the probability that the running total equals N at the end of round R.

Here, we denote a strategy for the game to be a function s that maps an index $i \in \{0, 1, 2, \ldots, R-1\}$ and an integer $t_i \in [-2i, 2i]$ to an action in $\{A, B, A \wedge B, \emptyset\}$; in other words, s makes an action that depends on the current round and the current running total.

The strategy s^* that we are going to design has the property that if $s^*(i, t_i) = X$, then in the case that the running total after the first i rounds is t_i, flipping coins according to action X maximizes the probability of getting N by the end of round R.

We will go through a few steps to design a dynamic programming algorithm to find the strategy s^*.

(a) Let $P[i, t]$ be the maximum probability that the running total is N at the end of round R **conditioned on** that the current running total is t after the first i rounds. What are the values of $P[R, N]$, $P[R-1, N]$, and $P[R-1, N-2]$?

(b) By definition of $P[i, t]$, the maximum probability of winning this game is $P[0, 0]$. To get $P[0, 0]$, we need to compute $P[i, t]$ for all $i \in \{0, \ldots, R\}$ and $t \in \{-2i, \ldots, 2i\}$. What are the base cases and how do we initialize them?

(c) Determine a recurrence relation for $P[i, t]$.

(d) Given the maximum probabilities $P[i, t]$ for all $i \in \{0, \ldots, R\}$ and $t \in \{-2i, \ldots, 2i\}$, how do we find the strategy $s^*(i, t)$?

5 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 2×1 shape pattern in any direction. **Clearly describe your algorithm, prove its correctness, and analyze its runtime as well as space complexity. Your runtime should be $O(2^{3L}LM)$. 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?

6 Coding Question

For this week's homework, you'll implement **Edit Distance** and **Global Alignment Construction** in the python jupyter notebook called `edit_distance.ipynb`. There are two ways you can access the notebook and complete the problems:

1. Click here and navigate to the `HW8` folder if you prefer to complete this question on Berkeley DataHub.

2. Run

   `git clone https://github.com/Berkeley-CS170/cs170-coding-notebooks-sp23`

   in your computer's terminal (and navigate to the `HW8` folder) if you prefer to complete it locally. If you run into any issues with python import or autograder errors, please refer to the local setup instructions here.

Notes:

- *Submission Instructions:* Please download your completed `edit_distance.ipynb` file and submit it to the gradescope assignment titled "Homework 8 Coding Portion".
- *OH/HWP Instructions:* There will be designated office hours for you to come to ask for conceptual and debugging help. Please visit the coding OH post for more information.
- *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.