AAD Question Bank - 4

Advanced Dynamic Programming

September 2025

Introduction

This problem set is ungraded and meant purely for practice. The problems have been shuffled and do not include topic tags, so that you can approach them without a bias. If you encounter any doubts, feel free to reach out during office hours. In case of typos or unclear statements, please let us know.

In the exam, you might be asked to "give an algorithm" to solve a problem. Your write-up should take the form of a short essay. Start by defining the problem you are solving and stating what your results are. Then provide:

- A description of the algorithm in English and, if helpful, pseudo-code.
- A proof (or proof sketch) for the correctness of the algorithm.
- An analysis of the running time.

We will give full credit only for correct solutions that are described clearly.

Problem 1

In a faraway kingdom, the royal chef is tasked with preparing a series of dishes for a grand feast. Each dish can either be savory (type 1) or sweet (type 2), and there are K serving platters available to present the dishes. The chef must carefully arrange the dishes onto the platters, while adhering to the following rules:

1. The order in which the dishes are served must be maintained. A dish that appears earlier in the sequence cannot be placed on a platter after a dish that appears later.
2. Every platter must have at least one dish, and all dishes must be placed on the platters.
3. For each platter, the chef calculates the *interaction score* by multiplying the number of savory dishes with the number of sweet dishes on that platter and summing this across all platters. The goal is to minimize the interaction score.

Example:
Dishes: $\{12112\}$, $K = 2$
The chef has 4 different ways to arrange the dishes onto the platters:

(a) $\{1\}, \{2112\}$: score $= 1 \times 0 + 2 \times 2 = 4$

(b) $\{12\}, \{112\}$: score $= 1 \times 1 + 2 \times 1 = 3$

(c) $\{121\}, \{12\}$: score $= 2 \times 1 + 1 \times 1 = 3$

(d) $\{1211\}, \{2\}$: score $= 3 \times 1 + 0 \times 1 = 3$

In this example, the minimum possible interaction score is 3.
Task:
Develop an algorithm to determine the minimum possible total interaction score, while adhering to the specified constraints. If it is not possible to serve the dishes on K platters while following the rules, return -1.

Problem 2

In a distant land, a traveler finds themselves lost in a dangerous landscape in the form of a 2D grid consisting of $M \times N$ zones. Each zone has different effects on the traveler's strength — some areas drain their strength, while others offer moments of relief, restoring their strength. The traveler begins their journey at the north-west (top left) point of the land and must reach the south-east (bottom right) point to find their way home.

The traveler can only move southward (down) or eastward (right) at each step of the journey. As they move through the zones, they lose or gain strength based on the value of the zone in the grid. (Negative values will decrease the strength, and Positive values will increase the strength, by the given value). The challenge is that the traveler's strength can never drop to zero or below, or they will not survive the journey.

The task is to find the minimum amount of initial energy needed to complete the journey.
Example: Consider the following map of zones, where each number represents the effect on the traveler's energy:

$$\begin{bmatrix} -2 & -4 & 15 \\ -1 & -5 & -4 \\ -4 & 20 & -1 \end{bmatrix}$$

If the traveler chooses the path `EAST` $\rightarrow$ `EAST` $\rightarrow$ `SOUTH` $\rightarrow$ `SOUTH`, the initial energy needed to complete the journey and reach home is 7. This is the minimum possible initial energy required.
Notes:

1. The traveler's energy has no maximum limit.
2. All zones in the landscape either drain energy or restore it, including the starting point (the top left zone) and the final destination (the bottom right zone).

Task:
Describe an algorithm to determine the minimum amount of energy the traveler must start with to ensure they can complete their journey safely.

Problem 3

We wish to transform a source string $x[1..m]$ into a target string $y[1..n]$ using a sequence of transformation operations. Each operation has an associated constant cost. The allowed operations are:

- **Copy:** set $z[j] = x[i]$, then increment i, j.
- **Replace:** set $z[j] = c$, then increment i, j.
- **Delete:** increment i only.
- **Insert:** set $z[j] = c$, increment j only.
- **Twiddle:** exchange next two characters $x[i], x[i+1]$ by copying them to z in reverse order.
- **Kill:** set $i = m + 1$ (must be final).

The cost of a transformation sequence is the sum of the costs of the chosen operations. The *edit distance* between x and y is the minimum possible cost to transform x into y.
Tasks:

(a) Describe a dynamic programming algorithm that computes the edit distance from $x[1..m]$ to $y[1..n]$ and prints an optimal sequence of operations. Analyze its running time and space complexity.

(b) Show how the sequence alignment problem in bioinformatics can be cast as a special case of this edit-distance formulation using a suitable subset of the operations.

Problem 4

Let $A[1..n]$ be an array of positive integers. An *increasing back-and-forth subsequence* is a sequence of indices $I[1..\ell]$ satisfying the following properties:

1. $1 \leq I[j] \leq n$ for all j.
2. $A[I[j]] < A[I[j+1]]$ for all $1 \leq j < \ell$.
3. If j is even, then $I[j+1] > I[j]$ (move right).
4. If j is odd, then $I[j+1] < I[j]$ (move left).

In less formal terms, imagine a token placed on any square of the array. At each step, the token moves to a square containing a larger number. The direction of movement alternates: left when on an odd-indexed step, right when on an even-indexed step. The sequence of positions visited by the token forms an increasing back-and-forth subsequence.
Task:
Design an algorithm to compute the length of the longest increasing back-and-forth subsequence for a given array $A[1..n]$.

Problem 5

Suppose we want to typeset a paragraph of text consisting of n words, where the i-th word has length ℓ_i. We want to break the paragraph into several lines of total length exactly L, inserting white space as needed.

The paragraph should be *fully justified*, meaning:

- The first character on each line starts at the left margin.
- Except for the last line, the last character on each line ends at the right margin.

There must be at least one unit of white space between any two words on the same line, and no space is needed after the last word on the line.

Define the *slop* of a paragraph layout as the sum over all lines, except the last, of the square of the amount of extra white space in each line beyond the mandatory one unit between adjacent words.

Specifically, if a line contains words i through j, then the total space used by mandatory spaces is $(j - i)$ and the total length of words is $\sum_{k=i}^{j} \ell_k$. The line satisfies:

$$(j - i) + \sum_{k=i}^{j} \ell_k \leq L,$$

and the slop for this line is

$$\text{slop}(i, j) = \big(L - (j - i) - \sum_{k=i}^{j} \ell_k\big)^2.$$

The slop of the paragraph is the sum of the slops for all lines except the last.

Design and analyze an efficient dynamic programming algorithm to break the paragraph into lines while minimizing the slop of the paragraph. Your algorithm must:

- Respect the word order.
- Ensure each line satisfies the length constraint.
- Minimize the sum of slops over all lines except the last.

Input: A list of word lengths $\ell_1, \ell_2, \ldots, \ell_n$ and line length L.
Output: The minimum possible slop of the paragraph.