C343-Data Structures
Assignment 3

Muazzam Siddiqui

This assignment has 5 problems. For each: (i) note what's provided in starter code, (ii) complete the tasks.

Problem 1: Autonomous Vehicle

In this problem, you will simulate an autonomous vehicle driving through a grid world containing various types of roads and obstacles. The vehicle must find a route from its starting position to a specified goal location while maximizing its overall reward.

Each cell in the grid may contain different conditions:

- **Two-lane roads:** Allow movement in both directions.
- **One-way roads:** Restrict movement to a single allowed direction.
- **Construction zones:** Impassable areas that must be avoided.
- **Crossings:** Special cells that may alter the path's reward.

The vehicle's total reward accumulates based on valid moves and is penalized by a constant `DISTANCE_PENALTY` for each additional step taken. The goal is to find the path that yields the highest total reward while obeying the movement constraints.

Your task

Following comments in starter code, implement `findBestPath` using recursive search.

Problem 2: Sequence Alignment

The sequence alignment problem comes from bioinformatics and text comparison. You are given two sequences (strings), say:

$$X = A\ C\ C\ T\ G\ A$$
$$Y = A\ C\ G\ T\ G\ A$$

and you want to align them by possibly inserting gaps (–) so that as many symbols as possible line up meaningfully. Example alignment:

$$\begin{array}{cccccc} A & C & C & T & G & A \\ | & | & & | & | & | \\ A & C & - & T & G & A \end{array}$$

The goal is usually to maximize similarity (matches) or minimize cost (mismatches + gaps).

Scoring Model

Typical scoring scheme: Match = +1, Mismatch = –1, Gap = –2.

The dynamic programming recurrence here, i.e. Needleman–Wunsch equations, is defined as:

$$S[i][j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0, \\ \max \begin{cases} S[i-1][j-1] + matchScore(X[i], Y[j]), \\ S[i-1][j] + gapPentality, \\ S[i][j-1] + gapPentality \end{cases} & \text{otherwise.} \end{cases}$$

where $matchScore(X[i], Y[j]) = +1$ if $X[i] = Y[j]$ (match), and -1 otherwise (mismatch).

Your task

Following comments in starter code, implement for `AlignmentResult` using the Needleman–Wunsch dynamic programming algorithm with gap penalties to compute one optimal global alignment, its score, and the aligned strings via backtracking.

Problem 3: Binomial Coefficient

The binomial coefficients can be defined recursively as follows:

$$\binom{n}{k} = \begin{cases} 1, & \text{if } k = 0 \text{ or } k = n, \\ \binom{n-1}{k-1} + \binom{n-1}{k}, & \text{for } 0 < k < n. \end{cases}$$

Your tasks

Following comments in starter code

1. Write a method to compute binomial coefficient for a given (n, k) pair
   - recursively
   - using dynamic programming
2. Give analysis of the running time comparing both methods

Problem 4: Circle Packing

The one-dimensional circle packing problem is as follows: You have N circles of radii $r_1, r_2, \ldots, r_N$. These circles are packed in a box such that each circle is tangent to the bottom of the box and are arranged in the original order. The problem is to find the width of the minimum-sized box. For example, with circles of radii $2, 1, 2$ respectively, the minimum-sized box has width $4 + 4\sqrt{2}$.

Your tasks

Following comments in starter code, implement `packWidth` to compute the minimal box width by greedy left-to-right placement.

Problem 5: Right Justifying a Paragraph

Problem

Consider the problem of right-justifying a paragraph. The paragraph contains a sequence of words $w_1, w_2, \ldots, w_n$ of length $a_1, a_2, \ldots, a_n$, which we wish to break into lines of length L), but blanks can stretch or shrink as necessary (but must be> 0)), so that a line $w_i, w_{i+1}, \ldots, w_j$ has length exactly L. However for each blank b' we charge $|b' - b|$ ugliness points. The exception to this is the last line, for which we charge only if $b' < b$ (in other words, we charge only for shrinking), since the last line does not need to be justified. Thus, if b_i is the length of the blank between a_i and a_{i+1}, then the ugliness of setting any line (but the last) $w_i, w_{i+1}, \ldots, w_j$ for $j > i$ is $\sum_{k=i}^{j-1} |b_k - b| = (j - i)|b' - b|$, where b' is the average size of a blank on this line. This is true of the last line only if $b' < b$, otherwise the last line is not ugly at all.

The goal is to use dynamic programming algorithm to find the least ugly setting of $w_1, w_2, \ldots, w_n$ into lines of length L. (Hint: For $i = N, N - 1 \ldots, 1$, compute the best way to set $w_i, w_{i+1}, \ldots, w_j$.)

Your task

Following comments in starter code, implement the `justify` method using dynamic programming to minimize the total ugliness, where ugliness is defined as the sum of absolute deviations of actual blank lengths from the ideal blank length b, following the given feasibility and cost rules.

How to Submit

1. Upload all the files listed to upload under the button "What files to submit" on Autograder to receive Autograder feedback.
2. Submit by pushing your work to your private github repo within the course organization.