Case study: Finding the winning strategy in a card game in python

Problem Description:

Imagine a card game where each player receives a hand of cards with values. The objective is to find the best way to maximize the score for a player, assuming the players take turns drawing cards. Each player can either pick the first or last card from the remaining pile.

Assumptions:

- Each player tries to maximize their score.
- Cards are represented by integers, which indicate their values.
- Two players alternate turns, and each player picks a card from either the beginning or the end of the list.

We need to design an algorithm that helps a player find the optimal strategy to guarantee the highest possible score given that the opponent is also playing optimally.

Plan:

We can solve this problem using **Dynamic Programming** by calculating the optimal score for every possible scenario, taking into account the best choices for both players.

Steps:

1. **Define the Game:** Represent the pile of cards as a list of integers.
2. **Recursive Strategy:** A function will recursively determine the best score a player can achieve.
3. **Dynamic Programming (Memoization):** Store intermediate results to avoid recalculating them.
4. **Base Cases:** When only one card is left, the current player takes it.

Program:

```
def find_optimal_strategy(cards):
  n = len(cards)
  # Create a memoization table to store subproblem results
  dp = [[0] * n for _ in range(n)]

  # Fill the table for subproblems of increasing sizes
  for length in range(1, n+1):
    for i in range(n-length+1):
```

```
        j = i + length - 1

        # If only one card is left, the player takes it
        if i == j:
            dp[i][j] = cards[i]
        else:
            # Choose the best of two choices:
            # 1. Take the left card, and the opponent plays optimally on the remaining (i+1, j)
            # 2. Take the right card, and the opponent plays optimally on the remaining (i, j-1)
            take_left = cards[i] - dp[i+1][j]
            take_right = cards[j] - dp[i][j-1]
            dp[i][j] = max(take_left, take_right)

    # dp[0][n-1] will have the optimal score difference for the first player
    return (dp[0][n-1] + sum(cards)) // 2  # First player's maximum possible score

# Example case
cards = [3, 9, 1, 2]
print("First player's optimal score:", find_optimal_strategy(cards))
```

Explanation:

- **Dynamic Programming Table (dp):** Each cell dp[i][j] represents the difference in score between the first player and the opponent if the game is played between cards from index i to index j.
- **Two Choices:** For each move, the player can either:
    1. Pick the leftmost card cards[i], leaving the opponent to play optimally on the remaining cards.
    2. Pick the rightmost card cards[j], leaving the opponent the rest of the cards.
- **Recursive Relation:** The value of each subproblem is determined by maximizing the score difference between the current player and the opponent.

Example Walkthrough:

Consider the array of cards: [3, 9, 1, 2].

1. **First player (you)** can choose between:
    - Taking the leftmost card (3), leaving the cards [9, 1, 2].
    - Taking the rightmost card (2), leaving the cards [3, 9, 1].
2. The opponent will then take their turn, playing optimally to minimize the first player's score.

This program computes the best possible outcome for the first player.

First player's optimal score: 5

First player, if playing optimally, can guarantee a score of 5 regardless of how the opponent plays.

Optimizing Strategy:

By using **Dynamic Programming**, we ensure that the solution is computed efficiently, avoiding redundant calculations. This approach ensures both players play optimally, and the first player gets the highest score possible given the opponent's best move.