Complete Documentation for the Solution

Table of Contents

1. Problem Statement
2. Intuition
3. Key Observations
4. Approach
5. Edge Cases
6. Complexity Analysis
    - Time Complexity
    - Space Complexity
7. Alternative Approaches
8. Test Cases
9. Final Thoughts

1. Problem Statement

In the "100 game", two players take turns adding numbers to a running total. Each player can choose a number from 1 to a given maxChoosableInteger. The goal is to force the opponent into a situation where they have no valid moves, or the total reaches or exceeds a target value (desiredTotal), causing them to lose.

We are tasked with determining if the first player to move can force a win given the constraints:

- No number may be reused.
- The player who causes the total to exceed or equal the target wins.

Given maxChoosableInteger (the maximum number a player can choose) and desiredTotal (the target score to win), the objective is to return True if the first player can force a win, otherwise return False. Both players play optimally.

2. Intuition

The game is essentially a combinatorial game where we must predict the outcome of every possible move given that both players play optimally. We use a recursive dynamic programming (DP) approach to explore all possible states of the game, using memoization to store already computed results to optimize the solution.

Key Concepts:

- State Representation: The game state can be represented by a bitmask where each bit indicates whether a number (from 1 to maxChoosableInteger) has been chosen.
- Recursion: The recursive function evaluates if the current player can force a win by making a move that leads to a winning situation.
- Memoization: To prevent redundant calculations, previously computed results are cached and reused.

3. Key Observations

- Total Sum: If the sum of all numbers from 1 to maxChoosableInteger is less than desiredTotal, the first player cannot win because even using all available numbers will not meet the target.
- Immediate Winning Condition: If the desiredTotal is 0, the first player wins automatically since the target has already been reached.
- Optimal Play: Both players play optimally, meaning each player will try to make the best move at every step, forcing the opponent into losing positions.
- Symmetry: The game is symmetric, meaning that if a player has a winning strategy, the opponent's moves are irrelevant except for their responses to force the game back into a favorable state.

4. Approach

Recursive Solution with Memoization:

i. Base Case:
    a. If the desiredTotal is 0 or less, the first player has already won.

b. If the sum of all numbers from 1 to maxChoosableInteger is less than desiredTotal, the first player loses automatically.

ii. Recursive Function:

a. Use a bitmask to represent the set of available numbers. Each player picks an available number, and the game state updates.

b. If a player's move results in a total greater than or equal to desiredTotal, they win immediately.

c. If not, the opponent plays optimally, so the current player checks if there's a way to force a win (i.e., any move where the opponent cannot win in the subsequent state).

iii. Memoization:

a. A dictionary is used to store the results of previously computed states to avoid redundant calculations and optimize performance.

Algorithm:

- Define the recursive function can_win(used, total), where used is the bitmask representing the chosen numbers, and total is the current total.
- For each number that has not been used, simulate the move and recursively check if the opponent can win. If the current player can find a move where the opponent cannot win, return True.
- Use memoization to cache results for different states (used).

5. Edge Cases

- Desired Total is 0: If desiredTotal == 0, the first player wins automatically because the total is already met.
- Impossible to Win: If the sum of all numbers from 1 to maxChoosableInteger is less than desiredTotal, it is impossible for the first player to win.
- Single Move Wins: If maxChoosableInteger >= desiredTotal, the first player can directly win by choosing the right number.
- Repetition of States: Since we are using memoization, duplicate states (i.e., states where the bitmask used is the same) will not lead to redundant calculations.

6. Complexity Analysis

Time Complexity:

- Time Complexity: The recursive function explores every possible combination of moves. The number of possible game states is 2^n where n is maxChoosableInteger. For each state, we check all available numbers (n choices), leading to a time complexity of $O(2^n * n)$.

Space Complexity:

- Space Complexity: The space complexity is mainly due to the memoization cache, which stores results for each state. The maximum number of states is 2^n, so the space complexity is $O(2^n)$.

7. Alternative Approaches

- Brute Force: One could try every possible sequence of moves for both players, but this would lead to exponential time complexity, making it impractical for larger values of maxChoosableInteger.
- Iterative DP: One could attempt an iterative dynamic programming approach, but this would still require tracking all possible states and transitions, leading to similar time and space complexity as the recursive approach with memoization.

8. Test Cases

i. Test Case 1:
   a. maxChoosableInteger = 10, desiredTotal = 11
   b. Expected Output: False
   c. Explanation: No matter which number the first player chooses, the second player will always win.

ii. Test Case 2:
   a. maxChoosableInteger = 10, desiredTotal = 0
   b. Expected Output: True
   c. Explanation: The first player wins immediately since the desired total is already reached.

iii. Test Case 3:

a. maxChoosableInteger = 10, desiredTotal = 1

b. Expected Output: True

c. Explanation: The first player can choose 1 to win immediately.

9. Final Thoughts

This solution efficiently solves the problem using dynamic programming with memoization. The recursive approach ensures that all possible states are explored, while memoization optimizes the solution by avoiding redundant calculations. Despite the complexity, the approach is efficient enough for the problem constraints, making it a practical solution for the problem.