2660. Determine the Winner of a Bowling Game

```
Simulation
Easy
      Array
```

Problem Description

turn. A complete game consists of n turns, and the maximum number of pins that can be knocked down in a single turn is 10. However, there's a special scoring rule at play. If a player hits all 10 pins (a strike) in a turn, then the score for any of the next two turns is doubled. More specifically, if x_i denotes the number of pins knocked down in turn i, then the score for turn i is:

In the given problem, we are simulating the scoring process of a bowling game for two players. Each player's progress in the

game is recorded in two separate arrays (player1 and player2), which represent the number of pins they knock down in each

• 2 * x_i, if the player scored a strike in either of the two preceding turns (i-1 or i-2).

• x_i, if the player did not score a strike in the previous two turns.

The task is to determine the winner of the game based on their scores; return 1 if player 1 has a higher score, 2 if player 2 has a

The total score for a player is the sum of the scores from all turns.

higher score, and 0 if there is a tie.

Intuition

To solve the problem, we can simulate the process of calculating each player's score turn by turn. We proceed in the following

manner:

2. During this process, we iterate through the array, and for each turn, we check whether a strike was hit in either of the two previous turns. 3. If a strike was hit in the previous two turns, we multiply the current turn's pins (x) by 2. Otherwise, we simply add the count of pins. 4. After we have iterated through all the turns and accumulated the score, we compare the scores of both players.

Solution Approach

1. We write a function f(arr) that calculates the total score for an array representing a player's knocked pins per turn.

The solution implements a function f(arr) which is used to calculate the scores for each player iteratively. The score calculation

logic reflects the special rule of the bowling game that states if a player scores a strike (10 pins) in one turn, the score for the

• For each turn, we check if there was a strike in any of the previous two turns. This is determined using the condition:

Here is a step-by-step approach to implementing the solution:

next two turns is doubled.

Iterate Through Turns: We loop through each turn in the array using enumerate, which gives us both the index (i.e., the turn number) and the score (i.e., the pins

(i and arr[i-1] == 10) or (i > 1 and arr[i-2] == 10)

and the score two turns back arr[i - 2] equals 10, we set k to 2; otherwise, k is 1.

• Finally, we compare the computed scores a and b to determine the winner. We return:

encapsulated within a single helper function, emphasizing clarity and maintainability.

Calculate Cumulative Score:

knocked down x) in that turn.

Check for Strikes in Previous Turns:

We calculate the score for the current turn by multiplying x (the pins knocked down) by k (the factor that may double the score if a strike occurred in one of the two previous turns) and add this to the cumulative score s. This step is repeated

∘ If the current turn index i is non-zero and the score in the immediate previous turn arr[i - 1] equals 10, or if the index is greater than 1

At the end of iterations, the function returns the total score s.

Compare Players' Scores:

■ 1 if a > b (player 1 wins),

■ 2 if b > a (player 2 wins), or

• 0 in case of a draw (equal scores).

for each turn in the array.

This solution takes a straightforward approach to simulating the scoring part of the game, avoiding the complexity of keeping track of strikes and multipliers outside of the immediate iteration. The code structure remains simple, and the logic is

Now, let's walk through how we would calculate the score for player1 using the function f(arr) as defined in the solution

player1 knocks down 1 pin. There was no strike in the previous turn (turn 3), but there was one in turn 2. Therefore, this

player2 knocks down 7 pins. No strike in the previous turn, so the score remains 7, and the cumulative total is 10.

player2 knocks down 2 pins. There was a strike in the previous turn, so this is doubled to 4, giving us a final total score for

• We invoke the scoring function f for both player1 and player2, storing the returned values in variables a and b, respectively.

Suppose we have the following pin arrays for two players: player1 = [4, 10, 5, 1]player2 = [3, 7, 10, 2]

Let's illustrate the solution approach with a small example:

Turn 1: player1 knocks down 4 pins. No previous turn, so score is 4.

Turn 3:

Turn 4:

Turn 1:

Turn 4:

Turn 3 (Strike):

• player1's score (a) is 26

• player2's score (b) is 24

from typing import List

Returns:

return 0

solution = Solution()

Example usage:

class Solution {

} else {

public:

total_points = 0

return total_points

if player1_total > player2_total:

if player2_total > player1_total:

return 1 # Player 1 wins

return 2 # Player 2 wins

Draw

Turn 2 (Strike):

approach:

Example Walkthrough

player1 knocks down 5 pins. There was a strike in the previous turn, so this score is doubled to 10, and the cumulative total now is 24.

player1 hits a strike with 10 pins. The score for this turn is 10.

score is also doubled to 2, giving us a final total score for player1 of 26.

player2 hits a strike with 10 pins. The score is 10, and the cumulative total is 20.

player2 knocks down 3 pins. The score is 3. Turn 2:

player2 of 24. Using the f(arr) function, we calculated the scores for both players: player1 has 26 points and player2 has 24 points. Comparing

Following the same process for player2:

Solution Implementation

def is_winner(self, player1_scores: List[int], player2_scores: List[int]) -> int:

Multiplies the score by 2 if the player scored a 10 in either of the two previous attempts.

If the current score is preceded by one or two scores of 10, the points for the current score are doubled.

multiplier = 2 if (i > 0 and scores[i - 1] == 10) or (i > 1 and scores[i - 2] == 10) else 1

Determines the winner based on the scores of player1 and player2.

int: 1 if player 1 wins, 2 if player 2 wins, or 0 for a draw.

def calculate_points(scores: List[int]) -> int:

total_points += multiplier * score

player1_total = calculate_points(player1_scores)

player2_total = calculate_points(player2_scores)

Determine the winner based on the total points calculated.

// Method to determine the winner based on the scores of both players

// Calculate the scores of player1 and player2 using the custom scoring fucntion

// Compare scores and return the result indicating the winner or a tie

// Helper method to calculate the score for a player's array of scores ('arr')

if $((i > 0 \&\& arr[i - 1] == 10) || (i > 1 \&\& arr[i - 2] == 10)) {$

player1_scores (List[int]): List of integers representing player 1's scores.

player2_scores (List[int]): List of integers representing player 2's scores.

Iterate through the player's scores to calculate the total points.

Define a nested function to calculate the total points for a player.

int: 1 if player 1 wins, 2 if player 2 wins, or 0 for a draw.

print(result) # The output would be either 1, 2, or 0 based on the scores.

def calculate_points(scores: List[int]) -> int:

total_points += multiplier * score

for i, score in enumerate(scores):

// If the previous one or two scores were 10, set multiplier to 2

// Calculate score for the current position and add to the total score

int isWinner(vector<int>& player1, vector<int>& player2) {

int scoreSum = 0; // Initialize total score sum to 0

// Determine the multiplier based on previous scores

// Iterate over all elements in the score array

int multiplier = 1; // Default multiplier

for (int i = 0; i < arr.size(); ++i) {</pre>

scoreSum += multiplier * arr[i];

// Return the total score for the player

// Function to calculate total score for a player

const calculateScore = (scores: number[]): number => {

int scorePlayer1 = calculateScore(player1);

int scorePlayer2 = calculateScore(player2);

} else if (scorePlayer2 > scorePlayer1) {

if (scorePlayer1 > scorePlayer2) {

return 1; // Player 1 wins

return 2; // Player 2 wins

return 0; // It's a tie

int calculateScore(vector<int>& arr) {

multiplier = 2;

return scoreSum;

let totalScore = 0;

for i, score in enumerate(scores):

Calculate total points for both players.

Define a nested function to calculate the total points for a player.

Iterate through the player's scores to calculate the total points.

Since a > b, player1 is the winner, and therefore the function should return 1.

these scores (a for player1 and b for player2), we determine the winner:

Args: player1_scores (List[int]): List of integers representing player 1's scores. player2_scores (List[int]): List of integers representing player 2's scores.

class Solution:

Python

```
# result = solution.is_winner([10, 2, 6], [5, 10, 10])
# print(result) # The output would be either 1, 2, or 0 based on the scores.
Java
class Solution {
    // Determines the winning player based on scores
    public int isWinner(int[] player1Scores, int[] player2Scores) {
       // Calculate the scores for both players
       int player1Score = calculateScore(player1Scores);
        int player2Score = calculateScore(player2Scores);
       // Return 1 if player 1 wins, 2 if player 2 wins, or 0 for a tie
       if (player1Score > player2Score) {
            return 1;
        } else if (player2Score > player1Score) {
            return 2;
        } else {
            return 0;
    // Calculates the score for a player based on the scoring rules
    private int calculateScore(int[] scores) {
        int totalScore = 0;
       // Iterate through the scores array
        for (int i = 0; i < scores.length; ++i) {
           // Determine the multiplier based on the previous scores
            int multiplier = 1;
            if ((i > 0 \& \& scores[i - 1] == 10) || (i > 1 \& \& scores[i - 2] == 10)) {
                multiplier = 2;
            // Update the total score
            totalScore += multiplier * scores[i];
       return totalScore;
C++
```

```
TypeScript
function isWinner(player1: number[], player2: number[]): number {
```

Args:

Returns:

return 0

solution = Solution()

Example usage:

total_points = 0

return total points

Draw

result = solution.is_winner([10, 2, 6], [5, 10, 10])

Time and Space Complexity

};

```
// Iterate over the scores to calculate the total
          for (let turn = 0; turn < scores.length; ++turn) {</pre>
              totalScore += scores[turn];
              // Check the previous or the one before the previous score for a perfect 10
              // If found, add the current score again as a bonus
              if ((turn && scores[turn - 1] === 10) || (turn > 1 && scores[turn - 2] === 10)) {
                  totalScore += scores[turn];
          return totalScore;
      };
      // Calculate total scores for both players
      const scorePlayer1 = calculateScore(player1);
      const scorePlayer2 = calculateScore(player2);
      // Determine the winner
      // If player1's score is higher, return 1; if player2's score is higher, return 2; if it's a tie, return 0.
      return scorePlayer1 > scorePlayer2 ? 1 : scorePlayer1 < scorePlayer2 ? 2 : 0;</pre>
from typing import List
class Solution:
   def is_winner(self, player1_scores: List[int], player2_scores: List[int]) -> int:
       Determines the winner based on the scores of player1 and player2.
       Multiplies the score by 2 if the player scored a 10 in either of the two previous attempts.
```

// It doubles the score for the current turn if the previous turn or the one before that was a perfect score.

```
# Calculate total points for both players.
player1_total = calculate_points(player1_scores)
player2_total = calculate_points(player2_scores)
# Determine the winner based on the total points calculated.
if player1_total > player2_total:
    return 1 # Player 1 wins
if player2_total > player1_total:
    return 2 # Player 2 wins
```

multiplier = 2 if (i > 0 and scores[i - 1] == 10) or (i > 1 and scores[i - 2] == 10) else 1

If the current score is preceded by one or two scores of 10, the points for the current score are doubled.

The time complexity of the given code is O(n), where n is the length of the array player1 or player2. This is because the function f(arr: List[int]) -> int iterates through each element of the input array exactly once.

The space complexity of the code is 0(1) as it uses a fixed amount of space. The variables s, a, b, along with a few others, do not depend on the size of the input and are only used to store single values or perform basic arithmetic operations no matter how large the input array is.