1927. Sum Game

**Problem Description** 

### Medium **Game Theory**

In this game, two players, Alice and Bob, take turns modifying a string of even length containing digits and '?' characters, with Alice going first. Each turn consists of replacing a single '?' with a digit between '0' and '9'. The game concludes when no '?' characters remain in the string.

The objective for Bob to win is to have the sum of the digits in the first half of the final string equal to the sum of the digits in the second half. In contrast, Alice aims for these sums to be unequal.

The task is to determine whether Alice, assuming optimal play from both players, will win the game.

ntuition

maximize their chances of winning. The two important factors analyzed here are the count of '?' characters (cnt1 and cnt2) in both halves of the string and the sums

This solution exploits the fact that each player will play optimally, meaning they will make the best possible move at each turn to

(s1 and s2) of known digits in each half at the start.

Observations to consider:

# 2. If the sums differ by an amount that isn't exactly half of 9 times the difference in '?' count between the two halves, Alice can enforce a win by

2 capturing the second observation.

strategically placing digits to prevent Bob from balancing the sums.

1. If the total count of '?' is odd, then Alice will always win because Bob will be the last to play, contributing to an imbalance.

The implementation checks for these two conditions to predict the winner, with the expression s1 - s2 != 9 \* (cnt2 - cnt1) //

optimal play.

By examining the parity of '?' and the sums' balance in relation to '?' distribution, the solution concludes who wins the game under

Solution Approach

## Here is a step-by-step breakdown of how the algorithm works: **Determine '?' Counts and Partial Sums:**

• The solution begins by splitting the string into two equal parts since the game's outcome depends on the comparison of these two

The solution approach leverages simple arithmetic and logical checks based on the initial configuration of the string.

segments. It counts the number of '?' characters in each half (denoted as cnt1 for the first half, and cnt2 for the second half).

**Check Parity of '?' Characters:** 

- Since players alternate turns, if the total number of '?' is odd, it implies Alice will always have one extra turn. The parity check (cnt1 + cnt2) % 2 == 1 asserts whether the number of '?' characters is odd, which would mean Alice wins.
- **Evaluate Sums Based on '?' Distribution:**

It calculates the sum of all known digits (ignoring '?') in each half (referred to as s1 for the first half, and s2 for the second half).

• If parity is not the determining factor, the algorithm moves on to evaluate whether the difference in known sums (s1 and s2) can be

### ∘ This is judged by s1 - s2 != 9 \* (cnt2 - cnt1) // 2. The rationale is that each '?' can ultimately contribute a value between 0 to 9 when replaced by a digit. Bob's goal to balance the sums can be thwarted if the difference between s1 and s2 cannot be matched by strategically

possible number to add to the sum for each '?').

placing numbers where '?' exist.

the sums from equalling.

balanced by the '?' characters.

**Return the Winner:** • The booleans returned by the checks outlined above directly feed into the final return statement. If any of the conditions is true, Alice wins

• Here, 9 \* (cnt2 - cnt1) // 2 gives the maximum potential difference that Bob can cause by manipulating '?' (because 9 is the maximum

∘ If the actual difference s1 - s2 does not equal half of the potential difference, it implies that Alice, by playing optimally, can always prevent

(True). Otherwise, the game outcome is in favor of Bob (False), meaning that he can balance the sums even with optimal play from Alice. The solution uses only basic data structures (integer variables for counting and summing) and relies mainly on arithmetic and

Let's illustrate the solution approach with a small example: Suppose we have a string s of even length: "5?2?". Alice and Bob will take turns replacing '?' with digits in order to achieve their

conditional checks, showcasing a direct and efficient approach to predict the game's outcome.

# The string is split into two equal halves: "5?" and "2?".

plays.

**Return the Winner:** 

Solution Implementation

**Example Walkthrough** 

objectives as described.

determine the winner.

• The counts of '?' are determined: cnt1 = 1 in the first half, and cnt2 = 1 in the second half.  $\circ$  The partial sums are calculated: s1 = 5 in the first half, and s2 = 2 in the second half (ignoring '?').

**Determine '?' Counts and Partial Sums:** 

**Check Parity of '?' Characters:** ○ We check if the total number of '?' is odd: (cnt1 + cnt2) % 2 == (1 + 1) % 2 == 0. In this case, it is even, so this condition does not

○ Since the parity is not odd, we evaluate whether the difference in sums can be balanced: s1 - s2 != 9 \* (cnt2 - cnt1) // 2, which

translates to 5 - 2 != 9 \* (1 - 1) // 2. This simplifies to 3 != 0, which is true. This means the difference in the current sums (s1 and s2) cannot be precisely offset by the '?'

**Evaluate Sums Based on '?' Distribution:** 

cause is also zero. ∘ Since the condition s1 - s2 != 9 \* (cnt2 - cnt1) // 2 is satisfied, it implies that Bob cannot make the sums equal regardless of how he

characters available because the difference in the number of '?' in each half is zero, so the maximum potential difference that Bob can

 The condition from step 3 ensures Alice's victory. So, the function would return True, indicating Alice wins the game with optimal play. By following the solution approach, you can see how the initial set-up of the game allows us to predict the winner by using simple

arithmetic and logical checks. In this example, even though Bob has an opportunity to place a number, he cannot balance the

sums, guaranteeing Alice's victory.

class Solution: def sumGame(self, num: str) -> bool: # Calculate the length of the string to divide it into two halves length = len(num)

### sum\_left = sum(int(digit) for digit in num[:length // 2] if digit != "?") # Calculate the sum of digits in the right half, ignoring '?' sum\_right = sum(int(digit) for digit in num[length // 2:] if digit != "?")

# Count occurrences of '?' in each half of the string

# Calculate the sum of digits in the left half, ignoring '?'

# There are two conditions that make the game start in a losing state:

# 1. The total number of '?' is odd, one player will always be at a disadvantage

# 2. The sum difference is not equal to half the 9 times the difference in '?' counts

# (since each '?' can contribute from 0 to 9 to the sum, the average contribution is 4.5)

# The function can now be called with an instance of the Solution class and a numerical string as an argument

return (count\_left + count\_right) % 2 == 1 or sum\_left - sum\_right != 9 \* (count\_right - count\_left) // 2

count\_left = num[:length // 2].count("?")

count\_right = num[length // 2:].count("?")

C++

public:

class Solution {

bool sumGame(string num) {

# Example usage:

# sol = Solution()

# result = sol.sumGame("026??")

**Python** 

```
# print(result) # Output will be either True or False based on the game's state
Java
class Solution {
    public boolean sumGame(String num) {
        int length = num.length();
        int questionMarksLeftHalf = 0, questionMarksRightHalf = 0;
        int sumLeftHalf = 0, sumRightHalf = 0;
        // Loop through the left half of the string
        for (int i = 0; i < length / 2; ++i) {</pre>
            if (num.charAt(i) == '?') {
                // Count question marks in the left half
                questionMarksLeftHalf++;
            } else {
                // Sum the digits in the left half
                sumLeftHalf += num.charAt(i) - '0';
       // Loop through the right half of the string
        for (int i = length / 2; i < length; ++i) {</pre>
            if (num.charAt(i) == '?') {
                // Count question marks in the right half
                questionMarksRightHalf++;
            } else {
                // Sum the digits in the right half
                sumRightHalf += num.charAt(i) - '0';
```

boolean isOddNumberOfQuestionMarks = (questionMarksLeftHalf + questionMarksRightHalf) % 2 == 1;

int countLeft = 0, countRight = 0; // Initialize question mark counters for both halves

// Check if the difference in sums is not equal to half the difference in counts of question marks st 9

boolean isSumDifferenceInvalid = sumLeftHalf - sumRightHalf != 9 \* (questionMarksRightHalf - questionMarksLeftHalf) / 2;

```
} else {
   // Add the numeric value to the sum of the left half
   sumLeft += num[i] - '0'; // Convert char to int
```

int length = num.size(); // Store the length of the num string

// Loop through the first half of the num string

// Loop through the second half of the num string

for (int i = 0; i < length / 2; ++i) {</pre>

for (let i = length / 2; i < length; ++i) {</pre>

// Increment counter for question marks in the right half

sumRight += numString[i].charCodeAt(0) - '0'.charCodeAt(0);

// Add digit value to total sum of the right half

# Calculate the length of the string to divide it into two halves

// Check if the total number of question marks is odd

# Count occurrences of '?' in each half of the string

# Calculate the sum of digits in the left half, ignoring '?'

sum\_left = sum(int(digit) for digit in num[:length // 2] if digit != "?")

count\_left = num[:length // 2].count("?")

count\_right = num[length // 2:].count("?")

if (numString[i] === '?') {

++questionMarksRight;

def sumGame(self, num: str) -> bool:

length = len(num)

if (num[i] == '?') {

countLeft++;

int sumLeft = 0, sumRight = 0; // Initialize sum counters for both halves

// Increase question mark count for the left half if encountered

// Check if the total number of question marks is odd

// Determine the result of the game by using the calculated booleans

return isOddNumberOfQuestionMarks || isSumDifferenceInvalid;

```
for (int i = length / 2; i < length; ++i) {</pre>
            if (num[i] == '?') {
                // Increase question mark count for the right half if encountered
                countRight++;
            } else {
                // Add the numeric value to the sum of the right half
                sumRight += num[i] - '0'; // Convert char to int
       // Check if the game cannot be made equal
       // If the total number of '?' is odd, or if the sum difference is not equal to the difference in '?' count multiplied by
       // then the game is not balanced and thus return true, else the game could end in a draw, so return false.
       return (countLeft + countRight) % 2 == 1 || (sumLeft - sumRight) != 9 * (countRight - countLeft) / 2;
};
TypeScript
function sumGame(numString: string): boolean {
    // Calculate the length of the input string
    const length = numString.length;
    // Initialize counters for question marks and sum of digits for both halves
    let questionMarksLeft = 0, questionMarksRight = 0, sumLeft = 0, sumRight = 0;
    // Iterate through the first half of the string
   for (let i = 0; i < length / 2; ++i) {
       if (numString[i] === '?') {
            // Increment counter for question marks in the left half
           ++questionMarksLeft;
       } else {
           // Add digit value to total sum of the left half
            sumLeft += numString[i].charCodeAt(0) - '0'.charCodeAt(0);
    // Iterate through the second half of the string
```

```
// or if the double of the difference in sums is not equal to nine times the difference in question marks count
// The game is Alice's win if any of these conditions is true
return (questionMarksLeft + questionMarksRight) % 2 === 1 ||
       2 * (sumLeft - sumRight) !== 9 * (questionMarksRight - questionMarksLeft);
```

class Solution:

# Example usage:

# sol = Solution()

# result = sol.sumGame("026??")

Time and Space Complexity

Regarding space complexity:

} else {

# Calculate the sum of digits in the right half, ignoring '?' sum\_right = sum(int(digit) for digit in num[length // 2:] if digit != "?") # There are two conditions that make the game start in a losing state: # 1. The total number of '?' is odd, one player will always be at a disadvantage # 2. The sum difference is not equal to half the 9 times the difference in '?' counts # (since each '?' can contribute from 0 to 9 to the sum, the average contribution is 4.5) return (count\_left + count\_right) % 2 == 1 or sum\_left - sum\_right != 9 \* (count\_right - count\_left) // 2 # The function can now be called with an instance of the Solution class and a numerical string as an argument

Calculating n: This is performed in constant time, so its complexity is 0(1). Counting '?': The count() function is called twice, once on each half of the string. Since the string is of length n, and it's

# print(result) # Output will be either True or False based on the game's state

both halves together.

are linear with respect to the length of the string. Hence, the time complexity is O(n).

The time complexity of the given Python code can be broken down into several parts:

- divided into two halves, each of length n/2, counting '?' on each half takes 0(n/2), resulting in a total complexity of 0(n) for Summing digits: Similar to counting, summation is performed twice, once on each half of the string, excluding '?'. This also
- takes O(n/2) for each half, summing up to O(n) for the whole string. The final return statement involves a comparison operation, which is done in constant time 0(1).
- Considering all parts, the overall time complexity of the function is dominated by the counting and summing operations, which

The additional space used by the function is for storing the temporary variables cnt1, cnt2, s1, and s2. These are constant size variables, and their space does not grow with the input size. Therefore, no matter how large n gets, the space used remains constant, leading to a space complexity of 0(1).