2180. Count Integers With Even Digit Sum

Problem Description

Math

Easy

Simulation

not even. The challenge is to compute the number of these integers efficiently, without having to check each number individually, since

The task is to find and count all the positive integers up to a given number num whose digits add up to an even sum. For instance,

if num is 13, the number 12 would be counted because its digits 1 + 2 equals 3, which is an odd total, and hence its digit sum is

that would be too slow for large values of num.

Intuition

Every group of ten numbers (0-9, 10-19, etc.) contains exactly five numbers with an even digit sum. This holds because within any such sequence, the sum of the digits changes in a predictable way when moving from one number to the next: the ones place increases by one until it resets at zero when it hits 10, and then the process repeats.

The solution takes advantage of patterns in numbers and their sums. The key observations to arrive at the solution approach are:

- By considering the tens place separately, the solution calculates how many complete tens are in the number num. Multiply that count by 5 (since there are five numbers with an even digit sum in each group of ten) to get a baseline count. Adjust the baseline count based on the remainder of num when divided by 10 (the last digit of num), and whether the sum of
- the tens digits is odd or even. The formula ans = num // 10 * 5 - 1 starts by calculating the baseline even digit sum count, then subtracts 1 to adjust for the starting point (as the sequence starts at 0, not 1).
- even or odd. Depending on this result and the value of the last digit of num, the final answer (ans) must be corrected. Solution Approach

Next, the code calculates the sum of the tens place digits to determine if the sum, including the ones place of the original num, is

The provided Python solution uses mathematical reasoning rather than brute force iteration over every number up to num. Here's

a detailed breakdown of the implementation: First, it calculates the number of complete tens in num—this tells us how many groups of ten we have to deal with. Each

It subtracts 1 from this count by doing ans = num // 10 * 5 - 1 because the sequence we are counting starts from 1 and not 0. For example, if num is 20, without the subtraction, we would count from 0 to 19, but we actually need to count from 1 to 20.

group of ten has exactly five numbers with an even digit sum, hence num // 10 * 5. This gives us a preliminary count.

- x, s = num // 10, 0while x: s += x % 10
- x //= 10This loops through each digit of num / 10 and adds it to s. The significance of this sum is to determine if the sum of all digits besides the last digit is odd or even, which influences the count adjustment in the subsequent step.
- Finally, the adjustment is done with the line:

accounts for whether the sum will turn even after including the last digit.

numbers with an even digit sum, this gives us 2 * 5 = 10 as our preliminary count.

To verify our method, we can manually count the numbers with even digit sums up to 23:

arithmetic operations that replace the need for any complex data structures.

Then the algorithm calculates the sum of the tens place digits with the snippet:

```
Here's what happens in this line:
   num % 10 gives us the last digit of num.
   The expression (s & 1) gives us 1 if the sum of tens place digits s is odd, and 0 if it's even.
```

By adding 2 and then subtracting (s & 1), we're effectively deciding whether to add 1 or 2 to the final ans. This

The >> 1 is a bitwise right shift which divides the number by 2, discarding the remainder. This is used to finally adjust the

answer correctly, accounting for the fact that only half of the adjustments (whether we add 1 or 2) result in an even total

Following the steps outlined in the solution approach:

Next, we determine the sum of the tens place digits:

∘ s & 1 will evaluate to 0 since 2 (the sum of tens place digits) is even.

The last digit of num (23) is 3 (23 % 10).

Finally, we make the adjustment:

• Count of even digit sum numbers: 11

Solution Implementation

def countEven(self, num: int) -> int:

even_numbers_count -= 1

tens_and_above //= 10

Python

class Solution:

digit sum.

Example Walkthrough

23 // 10 gives us 2.

ans += (num % 10 + 2 - (s & 1)) >> 1

To illustrate the solution approach, let's consider the number num = 23. We want to find how many positive integers less than or equal to 23 have a digit sum that is even.

We first calculate the number of complete tens in 23, which is 2 (from 23 // 10). Since each group of 10 has exactly five

The key algorithmic patterns used in this solution are mathematical counting and digit manipulation, along with efficient

 Since 2 is less than 10, the loop terminates after adding 2 to s. \circ The sum of the tens place digits is s = 2, which is even.

Now, we subtract 1 because we are counting from 1 rather than starting from 0. The count becomes 10 - 1 = 9.

- ∘ The correction (num % 10 + 2) >> 1 equals (3 + 2) >> 1, which simplifies to 5 >> 1 or, in other terms, 5 // 2, which equals 2. Since the tens place sum s is even, there's no need to further adjust the result for this case.
- Adding the correction to our count gives 9 + 2 = 11. Therefore, there are 11 numbers less than or equal to 23 with an even digit sum.
- The manual count confirms that our method yields the correct result.

• Even digit sum numbers: 2, 4, 6, 8, 11, 13, 15, 17, 20, 22, 24 (treating 24 as 2 and 4 since we're looking at numbers up to 23)

- # If 'num' is 58, then there would be (58 // 10) which is 5 groups of 10, # each group contributing 5 even numbers, hence 5 * 5 even numbers even_numbers_count = (num // 10) * 5
- # Now calculate the sum of the digits for the tens place and greater. # This is to help us determine if the last digit contributes an even or odd number. tens and above = num // 10sum of digits = 0 while tens and above:

additional even number can be included if the ones place is 1-9, otherwise it's 0-8.

Initial calculation: every group of 10 numbers contains exactly 5 even numbers

makes 'num' itself an even number, this will subtract one from the count.

Calculate the adjustment needed for the last digit contribution.

If the sum of digits (tens place and above) is even, then an

If the sum of digits is odd, then it's off by one either way.

The adjustment uses bitwise operation to quickly determine if

Apply the last digit adjustment to the even numbers count.

last_digit_adjustment = $((num % 10) + 2 - (sum_of_digits & 1)) >> 1$

// Calculate the sum of the digits in the quotient when num is divided by 10

// Check if the last digit in num contributes to an even or odd sum.

// This sum is used to determine if the last digit will result in an even total sum.

Adjusting the calculation if the sum of digits in 'num'

sum of digits += tens_and_above % 10

we need to add one more to the count.

even_numbers_count += last_digit_adjustment

// Initialize a counter for the even sum numbers

Return the final count of even numbers.

Java class Solution { // Method to count the number of even digit sum numbers up to a given number

return even_numbers_count

public int countEven(int num) {

int evenSumCount = 0;

int digitSum = 0;

return evenCount;

let digitSum = 0;

digitSum += x % 10;

digitSum += x % 10;

function countEven(num: number): number {

let evenCount = Math.floor(num / 10) * 5 - 1;

// if an additional even number should be counted

we need to add one more to the count.

even_numbers_count += last_digit_adjustment

Return the final count of even numbers.

return even_numbers_count

Time and Space Complexity

last_digit_adjustment = $((num % 10) + 2 - (sum_of_digits & 1)) >> 1$

Apply the last digit adjustment to the even numbers count.

complexity and space complexity of this algorithm are analyzed as follows:

evenCount += ((num % 10) + 2 - (digitSum & 1)) >> 1;

for (int x = num / 10; x > 0; x /= 10) {

evenCount += ((num % 10) + 2 - (digitSum & 1)) >> 1;

// Calculate the number of even tens before the given num

// Initialize the sum of digits variable for tens place

// Calculate the sum of digits in the tens place and above

for (let $x = Math.floor(num / 10); x; x = Math.floor(x / 10)) {$

// Check the last digit of num and the parity of digitSum to determine

// Return the total count of even numbers before and including num

```
// Loop through all numbers from 1 up to and including 'num'
        for (int currentNumber = 1; currentNumber <= num; ++currentNumber) {</pre>
            // Variable to store the sum of the digits of the current number
            int digitSum = 0;
            // Calculate the sum of the digits of 'currentNumber'
            for (int x = currentNumber; x > 0; x /= 10) {
                digitSum += x % 10; // Add the rightmost digit to 'digitSum'
            // If the digit sum is even, increment the even sum count
            if (digitSum % 2 == 0) {
                ++evenSumCount;
        // Return the total count of numbers with an even digit sum
        return evenSumCount;
C++
class Solution {
public:
    int countEven(int num) {
        // Calculate the number of even numbers by integer division by 10 and then multiply by 5.
        // Subtracted by 1 because the range is from 1 to num-1.
        int evenCount = (num / 10) * 5 - 1;
```

// If digitSum is even, then (num%10 + 2) contributes to having an odd total sum if num's last digit is even.

// If digitSum is odd, then (num%10 + 2) contributes to having an even total sum if num's last digit is even.

// The right shift by 1 at the end is essentially dividing by 2, which works to calculate the final adjustment.

return evenCount;

};

TypeScript

```
class Solution:
   def countEven(self, num: int) -> int:
       # Initial calculation: every group of 10 numbers contains exactly 5 even numbers
       # If 'num' is 58, then there would be (58 // 10) which is 5 groups of 10,
       # each group contributing 5 even numbers, hence 5 * 5 even numbers
       even_numbers_count = (num // 10) * 5
       # Adiusting the calculation if the sum of digits in 'num'
       # makes 'num' itself an even number, this will subtract one from the count.
       even numbers count -= 1
       # Now calculate the sum of the digits for the tens place and greater.
       # This is to help us determine if the last digit contributes an even or odd number.
       tens and above = num // 10
       sum of digits = 0
       while tens and above:
           sum of digits += tens_and_above % 10
            tens and above //= 10
       # Calculate the adjustment needed for the last digit contribution.
       # If the sum of digits (tens place and above) is even, then an
       # additional even number can be included if the ones place is 1-9, otherwise it's 0-8.
       # If the sum of digits is odd, then it's off by one either way.
       # The adjustment uses bitwise operation to quickly determine if
```

The given Python code implements an algorithm that counts how many integers from 1 to num have an even digit sum. The time

Time Complexity 1. The first line within the countEven method, ans = num // 10 * 5 - 1, involves a few constant-time arithmetic operations (integer division, multiplication, and subtraction) and therefore executes in 0(1) time.

3. Inside the loop, again only constant-time operations—modulus, integer division, and accumulation into s—are performed. 4. The last line outside the loop involves a few more constant-time arithmetic operations, including modulus, addition, bitwise AND, subtraction, and bit-shifting.

Overall, the most time-consuming part of the algorithm is the while-loop, so the time complexity is 0(log(num)).

2. The while-loop iterates over the number of digits in num // 10, which is proportional to the logarithm of num. Hence, the loop runs in

Space Complexity

O(log(num)) time.

- For space complexity:
- space. 2. No additional data structures that grow with the input size are used. 3. The space taken to store the intermediate results does not depend on the size of num, as it is only affected by the number of digits in num

1. The method uses a fixed number of integer variables ans, x, s, and those used for simple calculations. This occupies a constant amount of

which is logarithmic in size. Therefore, the space complexity of the algorithm is 0(1) as it requires a constant amount of space regardless of the input size.