1399. Count Largest Group

Problem Description

Hash Table

Easy

their digits. Then, you need to return the count of the groups with the largest size. A group's size is determined by how many numbers share the same digit sum. For example, if n is 13, the numbers 11 (1+1=2) and 20 (2+0=2) would belong to the same group with a digit sum of 2. If the group with a digit sum of 2 has the most numbers in it, you would include that in your count of the largest groups. Intuition

In this problem, you are given an integer n. Your task is to count how numbers from 1 to n can be grouped based on the sum of

To solve this problem, the approach is to create a map that will keep track of how many times each digit sum appears across the numbers from 1 to n. The key intuition is that instead of directly grouping numbers and then comparing sizes, we can efficiently tally counts for each possible digit sum using a map or dictionary.

2. For each number, we calculate the sum of its digits. 3. We record the digit sum in a counter (dictionary) by incrementing the corresponding digit sum's count.

found a larger group size.

1. We iterate through all numbers from 1 to n.

4. As we update the counter, we also keep track of the maximum count (size of the largest group) encountered so far. 5. If we encounter a digit sum with a count higher than our current maximum, we update the maximum and reset our answer to 1 since we have

Here's a detailed walk-through of the implementation:

i. This process continues in a while loop until i becomes 0.

We then update the Counter cnt by incrementing the count for the computed digit sum s.

Here's the process to arrive at the solution:

6. If we encounter a digit sum with a count equal to the maximum, we increment our answer because this is another group of the same largest size.

The final answer is the count of digit sum groups/counts that equal the maximum size we found throughout the iteration. The provided Python solution follows this intuition and keeps track of the digit sums using the Counter class from the

collections module, offering a clean and efficient way to manage the counts.

Solution Approach

The solution uses a hash table and some basic arithmetic operations to solve the problem. Specifically, it employs a Counter

object from Python's collections module, which is a subclass of dictionary designed to count hashable objects.

it has appeared. Two variables, ans and mx, are initialized to 0. ans will store the final answer (the number of groups with the largest size),

and mx will store the current maximum size encountered. We iterate over every number from 1 to n inclusive, using a for loop.

A Counter called cnt is initialized to store the frequency of each possible digit sum. It maps each sum to the number of times

adding it to an accumulator variable s. After each step, we update i to i // 10 which effectively removes the last digit from

Inside the loop, for each number i, we calculate the digit sum by repeatedly taking i % 10 (which gives the last digit) and

After updating cnt, we compare the updated count cnt[s] to the current mx. If the new count is greater, this means we have a new largest group size, so we update mx to cnt[s], and reset ans to 1 as this is the first group of this new size.

If the new count cnt[s] is equal to the current mx, then we have found another group of the current largest size, and we

increment ans by 1. After the loop has finished executing, the variable ans holds the number of groups that have the largest size, and we return this value.

Mathematically, given s as the sum of digits of a number i, the operation to update cnt can be represented as:

- cnt[s] += 1And the comparisons to determine if we found a new largest group size or another group of that size is:
- elif mx == cnt[s]: ans += 1

The provided code efficiently calculates the required group sizes and finds the number of groups with the largest size, adhering

We begin to iterate from 1 to 15. Firstly, we take number 1, the sum of its digits is 1. We update cnt[1] to be 1. Since mx <

Next, we consider number 2, the sum of its digits is 2. We update cnt[2] to be 1. Now mx is still 1, but ans is now 2, as we

Number 10 has a digit sum of 1, updating cnt[1] to 2. Now mx < cnt[1], so we set mx to 2 and ans to 1, since we have

• For 11, cnt[2] becomes 2, tying with the current maximum size. This increments ans to 2 because there's another group with the same

For Number 15, the digit sum is 6, which also ties with the max after updating cnt [6] to 2; ans is incremented to 6.

After considering all numbers from 1 to 15, we find there are 6 groups tied for the largest size, which is 2. The groups are the

Therefore, the final answer that would be returned is 6, representing the count of the largest groups by digit sum from 1 to 15.

Let's walk through a small example using n = 15 to illustrate the solution approach:

This process continues for numbers 3, 4, 5, 6, 7, 8, and 9, each forming a new group of size 1. ans is now 9.

found a larger group size (group of digit sum 1 is now the largest with size 2).

digit sums of 1, 2, 3, 4, 5, and 6.

from collections import Counter

max count = 0

def countLargestGroup(self, n: int) -> int:

digit sum counter = Counter()

num_groups_with_max_count = 0

current number = i

while current number:

public int countLargestGroup(int n) {

for (int i = 1; i <= n; ++i) {

// Loop through each number from 1 to n.

current number //= 10

Initialize a counter to keep track of the sum of digits

Calculate the sum of the digits of the current number

Increment the count for the group represented by this sum of digits

Update max count and num groups with max count based on the current sums

sum of digits += current_number % 10

if max count < digit sum counter[sum of digits]:</pre>

max count = digit sum counter[sum_of_digits]

elif max count == digit sum counter[sum_of_digits]:

// Initialize a count array to keep track of the frequency of sums.

// Calculate the sum of digits of the current number.

// If we've found a new maximum size for our digit groups.

} else if (largestGroupSize == sumCounts[digitSum]) {

// Return the number of groups that have the largest size.

// Initialize an array to store the count of each sum of digits

let maxCount = 0; // Variable to keep track of the maximum count of any sum

let sum = 0; // Variable to store sum of digits of the current number i

let numMaxGroups = 0; // Variable to count how many groups have this maximum count

largestGroupSize = sumCounts[digitSum]; // Update the largest group size.

// If this group size is the same as the current largest, increment the count.

largestGroupSizeCount = 1; // Reset the count to reflect this new largest group size.

for (int x = i; x; x /= 10) {

++largestGroupSizeCount;

// Increment the count of the found sum.

if (largestGroupSize < sumCounts[digitSum]) {</pre>

digitSum += x % 10;

++sumCounts[digitSum];

return largestGroupSizeCount;

function countLargestGroup(n: number): number {

// Iterate over each number from 1 to n

// Calculate the sum of digits of i

for (let i = 1: i <= n: ++i) {

sum += x % 10;

sumCounts[sum]++;

const sumCounts: number[] = new Array(40).fill(0);

for (let x = i; x > 0; $x = Math.floor(x / 10)) {$

// Check if the current sum has a new maximum count

// Increase the count of this sum in the 'sumCounts' array

digit_sum_counter[sum_of_digits] += 1

num groups with max count = 1

num_groups_with_max_count += 1

class Solution:

if mx < cnt[s]:</pre>

ans = 1

mx = cnt[s]

Example Walkthrough

to the solution approach described.

cnt[1], mx is now set to 1 and ans is set to 1.

size as the largest found so far. For 12, cnt[3] becomes 2, tying again, and ans goes up to 3.

Numbers 11, 12, and 13 have digit sums of 2, 3, and 4 respectively:

have two groups (for digit sums 1 and 2) with the same size, 1.

Initialize a Counter named cnt and two variables: ans set to 0 and mx also set to 0.

- For 13, cnt[4] becomes 2, tying again, and ans goes up to 4. For Number 14, the digit sum is 5, and updating cnt [5] to 2 ties with the current max; ans is incremented to 5.
- Solution Implementation **Python**

Initialize variables to keep track of the maximum count and the number of groups with that count

int[] sumFrequency = new int[40]; // The maximum sum for n digits can be 9 * 4 = 36, thus 40 is safe.

int largestGroupSizeCount = 0; // This will hold the count of groups with the largest size.

int maxGroupSize = 0; // This will hold the maximum size of any group encountered.

int sumOfDigits = 0; // This will hold the sum of digits of the current number.

Iterate over each integer from 1 to n for i in range(1, n + 1): sum of digits = 0

Return the number of groups that have the maximum size return num_groups_with_max_count

Java

class Solution {

```
// Loop to calculate the sum of digits of the current number.
            for (int x = i; x > 0; x /= 10) {
                sumOfDigits += x % 10;
            // Increase the frequency count of the current sum.
            ++sumFrequency[sumOfDigits];
            // If the current sum frequency is greater than the maxGroupSize, update maxGroupSize and reset largestGroupSizeCount.
            if (maxGroupSize < sumFrequency[sumOfDigits]) {</pre>
                maxGroupSize = sumFrequency[sumOfDigits];
                largestGroupSizeCount = 1:
            } else if (maxGroupSize == sumFrequency[sumOfDigits]) {
                // If the current sum frequency is equal to the maxGroupSize, increment the count.
                ++largestGroupSizeCount;
        // Return the count of the sums that have the largest group size.
        return largestGroupSizeCount;
C++
class Solution {
public:
    int countLargestGroup(int n) {
        // Array to store counts of sum groups with a large enough size
        // to avoid the constant reallocation.
        int sumCounts[40] = {};
        // Variables to keep track of the largest group size and the
        // number of aroups with this size.
        int largestGroupSizeCount = 0;
        int largestGroupSize = 0;
        // Iterate over all numbers from 1 to n.
        for (int i = 1; i <= n; ++i) {
            int digitSum = 0; // This will hold the sum of the digits.
```

if (maxCount < sumCounts[sum]) {</pre> maxCount = sumCounts[sum]; // Update the max count numMaxGroups = 1; // Reset the number of max groups as a new max is found } else if (maxCount === sumCounts[sum]) {

TypeScript

```
// If the current sum is equal to the maximum count,
            // increment the number of groups with max count
           numMaxGroups++;
    // Return the number of groups with the maximum count of sums
    return numMaxGroups;
from collections import Counter
class Solution:
    def countLargestGroup(self, n: int) -> int:
        # Initialize a counter to keep track of the sum of digits
        digit sum counter = Counter()
        # Initialize variables to keep track of the maximum count and the number of groups with that count
        max count = 0
        num_groups_with_max_count = 0
        # Iterate over each integer from 1 to n
        for i in range(1, n + 1):
            sum of digits = 0
           current number = i
           # Calculate the sum of the digits of the current number
           while current number:
                sum of digits += current_number % 10
                current number //= 10
           # Increment the count for the group represented by this sum of digits
            digit_sum_counter[sum_of_digits] += 1
           # Update max count and num groups with max count based on the current sums
            if max count < digit sum counter[sum of digits]:</pre>
                max count = digit sum counter[sum_of_digits]
                num groups with max count = 1
            elif max count == digit sum counter[sum_of_digits]:
                num_groups_with_max_count += 1
        # Return the number of groups that have the maximum size
        return num_groups_with_max_count
```

The given Python code computes the size of the largest group of numbers based on the sum of the digits of each number from 1

To analyze the time complexity, let's examine the primary operations in the code: The for loop runs from 1 to n which gives us O(n) complexity. Inside the loop, we calculate the sum of digits of an integer i. Since in the worst case scenario integer i can have at most

each number.

Time and Space Complexity

to n.

Time Complexity:

Updating the counter for sum s and comparing mx with cnt[s] for each number is done in constant time, O(1).

The overall time complexity is thus the product of the complexity of the outer loop and the inner while loop, which gives us 0(n log n).

The variables s, i, mx, and ans use constant space O(1).

Space Complexity:

The counter cnt will at most have an entry for each distinct sum of digits s that we can get from numbers 1 to n. In the

O(log n) digits (since every increase by an order of magnitude adds one digit), the inner while loop runs in O(log n) time for

worst case, this sum would not exceed 9 * log n (assuming each digit of a number n is 9). Therefore, the space complexity due to the cnt map is O(log n).

For space complexity, we need to consider:

The space complexity of the algorithm is mainly determined by the cnt map. Hence, the overall space complexity is 0(log n). The code you provided is already efficient with respect to big-O notation, and any further optimization would depend on specific

constraints or requirements of the problem not detailed here.